
بسم الله الرحمن الرحيم

محمد عبد الوهاب منزلى

رحمه الله وغفر له

قال الله تعالى

بسم الله الرحمن الرحيم

يَا أَيَّتُهَا النَّفْسُ الْمُطْمَئِنَّةُ ارْجِعِي إِلَى رَبِّكِ رَاضِيَةً مَرْضِيَّةً فَادْخُلِي فِي عِبَادِي وَادْخُلِي جَنَّتِي

صدق الله العظيم

إلى من سكنت روحه أرواحنا ..إلى حبيب سكن قلوبنا

إلى من فارقت روحه عالمنا ..إلى من ذهب عنا ولم يودعنا

نسأل الله سبحانه وتعالى له الرحمة والمغفرة
ونتقدم لإسرته الكريمة وذويه بخالص العزاء والمواساة

ورحم الله من يهدي له ثواب الفاتحة

. بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ
الْحَمْدُ للَّهِ رَبِّ الْعَالَمِينَ (2) الرَّحْمَنِ الرَّحِيمِ (1)
مَالِكِ يَوْمِ الدِّينِ (4) إِيَّاكَ نَعْبُدُ وإِيَّاكَ نَسْتَعِينُ (5) اهدِنَا (3)
الصِّرَاطَ المُستَقِيمَ (6) صِرَاطَ الَّذِينَ أَنعَمتَ عَلَيهِمْ غَيرِ المَغضُوبِ
(عَلَيهِمْ وَلاَ الضَّالِّينَ (7

صدق الله العظيم

دفعه 2016 هندسه أسوان

Aswan University

Aswan Faculty of Eng.

Electrical Eng. Dep.

Computer and System Section

Egyptian License Plate Recognition

Supervisors:

Dr.Saleh Kamal Haridy

Dr.Mahmoud Saber

Teaching Assistants:

Eng.El-Sayed Abd-Ellah

Eng.Fatma Sayed

Team Work:

1. Eman Ahmed Muhammed
2. Eman Salem Saad Allah
3. Abeer Meshaal Abd-El Rehim
4. Fatma El-Zahraa Gamal
5. Fatma El-Zahraa Omar
6. Yousra Hesham Hassan

Acknowledgments

We thank ALLAH for his grateful gaudiness in every step in our life, and for helping us to complete this project.

We would like to thank our supervisors Dr. Saleh Kamal Haridy and Dr. Mahmoud Saber for their help and our research assistants Eng. EL-Sayad Abd-El lah and Eng. Fatma Sayad.

We also would like to thank everyone help us in this project.

Abstract

License Plate Recognition or LPR is an image-processing technology used to identify vehicles by their license plates. This technology is used in various applications involving security, traffic, law enforcement, public safety and transportation sectors. It mainly uses software code that enables computer systems to read automatically the registration number (license number) of vehicles from digital pictures.

Surveillance systems which depend only on human monitoring are likely to have many troubles, so we need to build an automatic system to solve this problem. In addition, traffic violations have been reported as a hard and tough task and it can be easily solved using an automatic license plate recognition system. The automatic license plate recognition system solve the problem of recognizing the Arabic numbers and letters which are written in the car license plate.

In this project, a high resolution camera is employed to capture images from streets and by applying various image processing techniques we can automatically detect license plates of the cars. The characters included in detected license plate are then recognized to decide the identity of each character.

We build a software system to detect, analyze the high resolution images which are captured by high resolution IP camera fixed on front of a gate or in street. This software system can automatically detect and recognize the Arabic letters and numbers and identify them. The algorithm employed in this project has the following steps: First, a surveillance camera will capture a video or a high resolution still image for the car (this image is the input for the next step). The input image will be then processed to find the location of the license plate. The letters and the numbers written on the detected plate sub image are segmented using fast segmentation technique. Finally, these segmented parts are recognized using pattern matching technique and the output be the number and letters of the car License plate.

We try a lot of algorithms for detection, segmentation and recognition. Some of these algorithms give good result and others not working well. In this book, we will present to you an overview of our work and the algorithms we use.

List of Contents

Chapter 1

Introduction

License Plate Recognition or LPR is an image processing technique which takes images of license plates and converts these images to text. While the idea may seem simple, but there is a big complexity goes on behind the scenes which is quite amazing . LPR is also called in different references as :

- Automatic Vehicle Identification (AVI)
- Car Plate Recognition (CPR)
- Automatic Number Plate Recognition (ANPR)
- Car Plate Reader (CPR)
- Optical Character Recognition (OCR) for Cars

1.1 History of License Plate Recognition

England has the highest CCTV camera deployment rate per government municipality and commercial enterprise, with estimated calculations showing nearly five million CCTV cameras currently in use. Not surprisingly, license plate recognition has its roots in the UK. The British Police Scientific Development Branch first invented License Plate Recognition Technology in 1976.

Earlier prototypes were premature with low accuracy readings and only functioned under restrictive laboratory conditions that made real world application nearly impractical.

Moderate improvements were made and working models were implemented a few years later in Wokingham, England. This subsequently led to the successful development of commercial contracts and applications for early adopter usage. While still crude, with an accuracy that averaged below 60%, the technology was state of the art during its time.

Several decades later, the technology has greatly evolved and prior limitations incurred from vehicle speed, light fluctuation, angular skew, character segmentation and recognition have been solved with today's algorithm technology. Additionally, prior cost prohibitive fees have made way for more reasonably priced implementations allowing application to thrive in multiple industries.

1.2 LPR Applications

LPR technique has a wide range of applications, which use the extracted plate number and optional images to create automated solutions for various problems. These include the following applications:

1.Parking - the plate number is used to automatically enter pre-paid members and calculate parking fee for non-members (by comparing the exit and entry times).
The optional driver face image can be used to prevent car hijacking .

Fig 1.1 parking example

In this example, a car is entering a car park in a busy shopping center. The car plate is recognized and stored. When the car will later exit (through the gate on the right side) the car plate will be read again. The driver will be charged for the duration of the parking.
The gate will automatically open after payment - or if the vehicle has a monthly permit.

2.Access Control - a gate automatically opens for authorized members in a secured area, thus replacing or assisting the security guard. The events are logged on a database and could be used to search the history of events .In this example, the gate has just been automatically raised for the authorized vehicle, after being recognized by the system. A large outdoor display greets the driver. The event (result, time and image) is logged in the database.

Fig 1.2 Access control

3.Tolling - the car number is used to calculate the travel fee in a toll-road, or used to double-check the ticket .
In this installation, the plate is read when the vehicle enters the toll lane and presents a pass card. The information of the vehicle is retrieved from the database and compared against the pass information. In case of fraud the operator is notified.

Fig 1.3 Tolling

4.Border Control - the car number is registered in the entry or exits to the Country, and used to monitor the border crossings. It can short the border crossing turnaround time and cut short the typical long lines.
This installation covers the borders of the entire Country. Each vehicle is registered into a central database and linked to additional information such as the passport data. This is used to track all border crossings.

5.Stolen cars - a list of stolen cars or unpaid fines is used to alert on a passing cars. The 'black list' can be updated in real time and provide immediate alarm to the police force. The LPR system is deployed on the roadside, and performs a real-time match between the passing cars and the list. When a match is found a siren or display is activated and the police officer is notified with the detected car and the reasons for stopping the car.

Fig 1.4 Border Control

6. **Enforcement -** the plate number is used to produce a violation fine on speed or red-light systems The manual process of preparing a violation fine is replaced by an automated process which reduces the overhead and turnaround time. The fines can be viewed and paid on-line . The image

Fig 1.5 Enforcement

Is an example of a speeding car caught by the traffic camera . The rear vehicle plate is automatically extracted off the scanned film image, replacing a tedious manual operation and the need to developed and print the violation. The data block on the top-right side is additional speeding information that is automatically extracted from the developed film and used to complete the fine notice and inserted to a database. The violators can pay the fine on-line and are presented with this image as a proof with the speeding information.

7. **Traffic control** - the vehicles can be directed to different lanes according to their entry permits (such as in University complex projects). The system effectively reduces traffic congestions and the number of attendants .In this installation the LPR based system classifies the cars on a congested entrance to 3 types (authorized, known visitors, and unknown cars for inquiry) and guides them to the appropriate lane. This system reduced the long waiting lines and simplified the security officers work load.

Fig 1.6 Traffic Control

8.Marketing Tool - the car plates may be used to compile a list of frequent visitors for marketing purposes, or to build a traffic profile (such as the frequency of entry verses the hour or day).

9.Travel - A number of LPR units are installed in different locations in city routes and the passing vehicle plate numbers are matched between the points.

The average speed and travel time between these points can be calculated and presented in order to monitor municipal traffic loads.

Additionally, the average speed may be used to issue a speeding ticket .

In this example the car is recognized at two points, and the violation shows the images of both locations

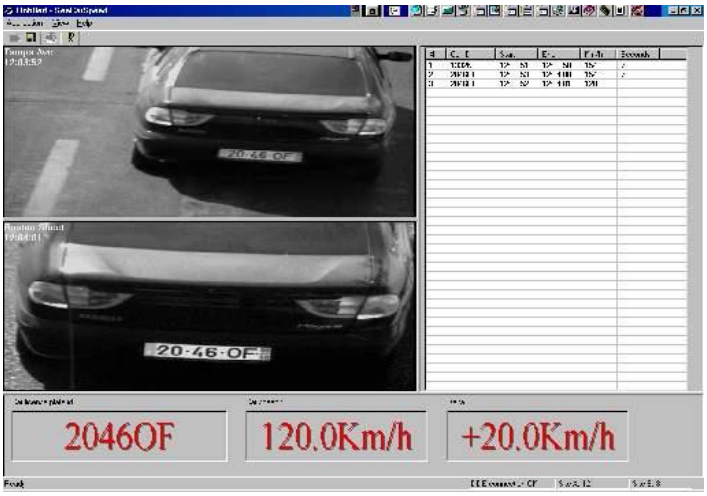

Fig 1.7 Travel

Which were taken on bridges on top of the highway. The average speed of the car is calculated from both points, and displayed if the speed passed a violation threshold, and optionally printed.

10. Airport Parking - In order to reduce ticket fraud or mistakes, the LPR unit is used to capture the plate number and image of the cars. The information may be used to calculate the parking time or provide a proof of parking in case of a lost ticket - a typical problem in airport parking which have relatively long and expensive parking durations .

This image shows the gate of a long term airport parking. The car is recognized on entry and the data is later used to track the real entry time in case of a lost ticket. The data is later used to track the real entry time in case of a lost ticket.

Fig 1.8 AirPort Parking

1.3 Egyptian license plate

Fig 1.8 Egyptian license plate

New Egyptian license plate has a fixed size 17 cm × 32 cm, 27 alphanumeric characters (17 alphabets and 10 numerical). The plate region is divided into three parts, the first part is high part of plate region with 62 mm that contains word of Egypt by Arabic and English.

Fig 1.9 Top part of Egyptian license plate

A background color of this region refers to type of car (private, taxi, etc.).

لون اللوحات	نوع المركبة
السماوى – الأزرق الفاتح	السيارات الملاكى
البرتقالى	السيارات الاجرة
الأزرق الغامق	سيارات الشرطه
الأحمر	سيارات النقل والجرارات
البنى	السيارات التجاريه
الأصفر	سيارات الجمارك
اللأخضر	سيارات الهيئة السياسية

Table 1.1 Background colors of top part of Egyptian license plate

The reminder region of the plate is divided vertically into two regions; right half contains plate characters, and left half contains numbers.

Fig 1.10 the bottom part of The Egyptian License plate

Now we will explain the means of characters and numbers in plate:

- First Cairo has 3 numbers and 3 letters, these numbers and letters isn't special, that means any letter or number can be in the plate, Giza also the same as Cairo but it has 4 numbers and 2 letters.
- Some Governorates has a special letter that the plate begin with it. (see table 1.2)

المحافظة	الحروف
الإسكندرية	س
الشرقية	ر
الدقهلية	د
البحيرة	ب
الغربية	غ
المنوفية	م
القليوبية	ق
كفر الشيخ	ك
الفيوم	ف

Table 1.2 Governorates with special character

- Canal governorates, Red Sea and North and South Sinai has a special first character (ط), the second character represent the conservatism the car follow and the third character the Traffic management has the full freedom to choose it, avoiding repetition. (table 1.3)

المحافظة	الحروف
بورسعيد	ط ع
السويس	ط س
دمياط	ط د
البحر الأحمر	ط ر
شمال سيناء	ط أ
جنوب سيناء	ط ج

Table 1.3 Governorates with same first character (ط)

- The two governorates including the new Valley and Matrouh both has same first character (ج) and a different second one. (Table 1.4)

المحافظة	الحروف
الوادي الجديد	**ج ب**
مطروح	**ج ه**

Table 1.4 Governorates with same first character (ج)

- Qena, Aswan and Luxor all start with (ص), the second character private to each governorates and the third the Traffic Management choose it with no repeat. (Table 1.5)

المحافظة	الحروف
قنا	**ص أ**
الاقصر	**ص ق**
أسوان	**ص و**

Table 1.5 Governorates with special two character

Chapter 2

Proposed system

In the previous chapter, we have a little introduction and small overview about LPR technique and its various applications. In this chapter, we will give a detailed explanation about our project and we show how a system with LPR work. Our project can be used on the Access Control Application.

2.1 LPR on Access Control Application

License plate recognition involves capturing image graphic video or images of license plates, whereby they are processed by a series of algorithms that are able to provide an alpha numeric conversion of the captured license plate images into a text entry .

2.1.1 How a typical access-control system works?

1. The vehicle approached the secured area, and starts the cycle by stepping over a magnetic loop detector (which is the most popular vehicle sensor). The loop detector senses the car and its presence is signaled to the LPR unit.

Fig 2.1

2. The LPR unit activates the illumination (invisible Infra-red in most cases) and takes pictures of the front or rear plates from the LPR camera.

Fig 2.2

3. The LPR unit analyzes the image with different image processing software algorithms, enhances the image, detects the plate position, extracts the plate string, and identifies the fonts using special artificial intelligence methods.
4. The LPR unit checks if the vehicle appears on a predefined list of authorized cars, and if found, it signals to open the gate by activating its relay. The unit can also switch on a green "go-ahead" light or red "stop" light. The unit can also display a Welcome! Message with personalized data.

Fig 2.3

5. The authorized vehicle enters into the secured area. After passing the gate its detector closes the gate. Now the system waits for the next vehicle to approach the secured area

Fig 2.4

2.2 Our real proposed system

The system is implemented at Aswan Faculty of engineering and it composed of a manual gate (Fig 2.5) and surveillance camera (Fig 2.6) linked to a pc via POE Ethernet switch (Fig 2.8), this pc has our application (Fig 2.9), this application takes the image from camera then use image processing to detect the plate, segment the characters and finally recognized the characters.

Fig 2.5 Faculty Gate

Fig 2.6 surveillance camera

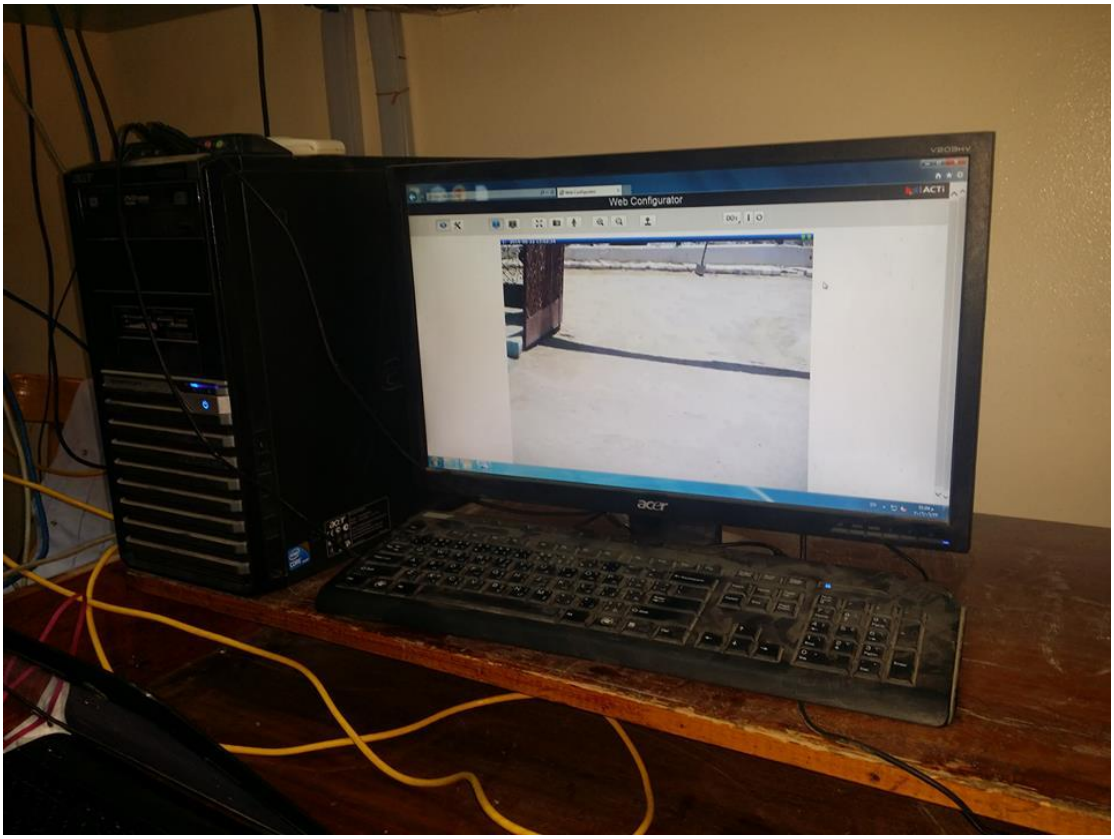

Fig 2.7 PC with Camera web application

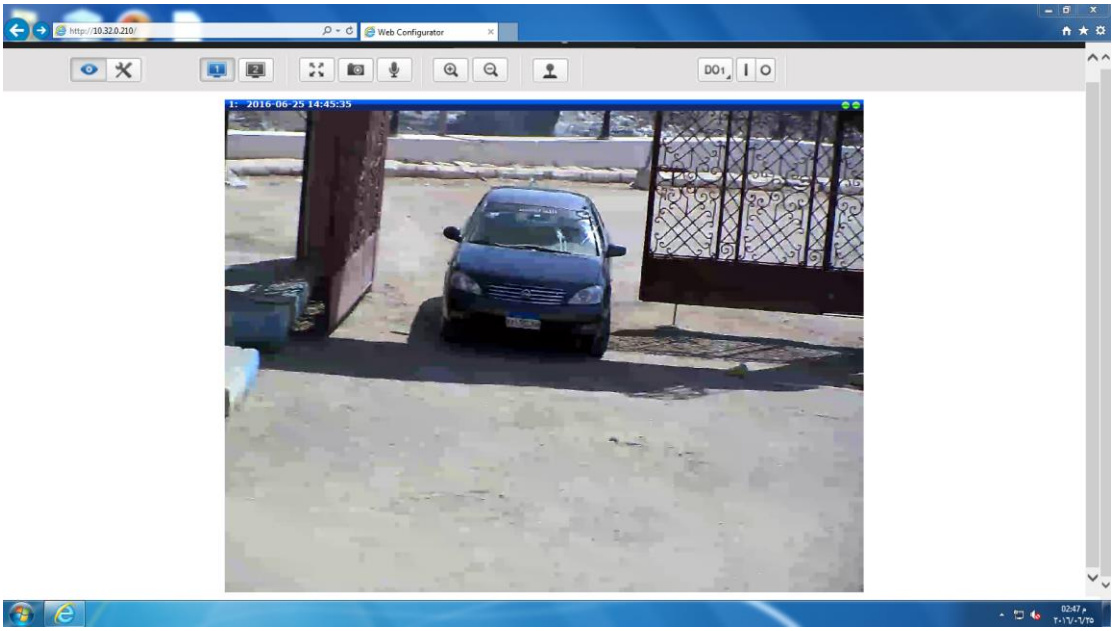

Fig 2.8 Camera Web interface screen shot

When the car reaches the faculty gate and be on the shadow line in the ground, we capture the image to the car from the pc and then enter this image to the application to recognize the license of this plate.

Figure 2.9 show you the block diagram of the system.

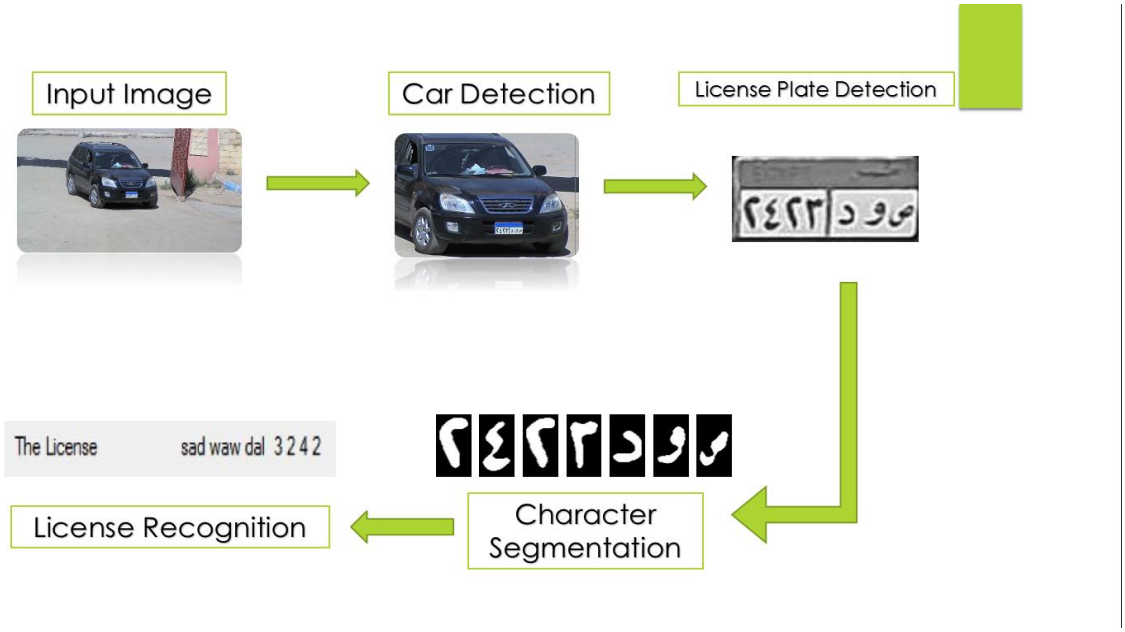

Fig 2.9 block diagram of proposed system

From this diagram, we can see the stages that the system goes through to get the license of the plate, these stages are:

1. A car reaches the gate, crossing the shadow line in ground.
2. We take a image to the car by the surveillance camera.
3. This image is given to our application.
4. After the application read it, Gaussian filter takes its turn to filter the image.
5. After filtering, it enters the detection stage where the detection algorithm tries detect the location of the plate in image.
6. Then come the stage of cropping the plate from image.
7. After that we segment the characters from the plate by a specific segmentation algorithm
8. Then come the stage of recognition this stage is responsible for recognize the segmented characters and convert it to text to get the plate.

2.3 ACTi B47 Camera

ACTi B47 camera has some specifications for which that we need to reach to a good result from our application. These specifications are:

- Maximum resolution 3MP
- Day/Night
- Adaptive IR LED
- Lens:
  - Zoom,

Fig 2.10 ACTi B47

- f5.2-62.4mm / F1.8-3.0,
- DC iris,
- Auto focus

- Maximum Frame Rate (30 fps) at 1920 x 1080

2.4 Software Development

The software application passes throw some steps for detecting the license.

2.4.1 Car detection

Before we begin to detect the car plate from the input image we detect the car only from the overall image because complex image with a lot of details will cause error detection output. There are many algorithms working on car detection

1. Haar cascaded classifier algorithm
2. Background Subtraction (BS).

2.4.2 Plate Detection

Plate detection is the first step in a LPR system. This step is for locating the number plate in the captured image. A number of techniques can be used in this step e.g. color detection, signature analysis, edge detection etc. Any tilt in the captured image is corrected in this step. Once the number plate is located the image is ready for character recognition.

2.4.3 Plate Segmentation

Segmentation is the stage where the characters are separated this can be done by detecting the transition from dark to light or from light to dark level. Each character present in the number plate produces a band of gray level. So by detecting the similar gray level bands each character can be segmented.

2.4.4 Recognition

This is the OCR step. The techniques that can be used in this step include pattern matching, feature matching.

In the following chapter, we give more details about each step in our proposed system.

Chapter 3

Detection and Segmentation

As we mentioned earlier license plate detection is the first step in a LPR system The goal of this step is to locating the plate and then to segment all numbers in the captured image.

The input to this stage is a car image, and the output is a portion of the image containing the potential license plate.

The license plate can exist anywhere in the image. Instead of processing every pixel in the image, which increases the processing time, the license plate can be distinguished by its features, and therefore the system processes only the pixels that have these features.

License plate features are:

1. License plate color
   Since some authorities (i.e., countries, states, or provinces) have certain colors for their license plates.
2. The rectangular shape of the plate
3. The texture
   The color change between the characters and the license plate background**.**
4. The existence of the characters can be used as a feature to identify the region of the license plate.

3.1 Detection Algorithms

We categorize the existing license plate extraction algorithms based on the features they used**.**

3.1.1 License Plate Extraction Using Global Image Information

This algorithm can be performed by three ways. These ways are as the following.

a. Connected component analysis (CCA) is an important technique in binary image processing it scans a binary image and labels its pixels into components based on pixel connectivity. Spatial measurements, such as area and aspect ratio, are commonly used for license plate extraction.

b. Contour detection algorithm is applied on the binary image to detect connected objects. The connected objects that have the same geometrical features as the plate are chosen to be candidates. This algorithm can fail in the case of bad quality images, which results in distorted contours.

c. 2-D cross correlation is used to find license plates. The 2-D cross correlation with a pre-stored license plate template is performed through the entire image to locate the most likely license plate area. Extracting license plates using correlation with a template is independent of the license plate position in the image.

3.1.2 License Plate Extraction Using Texture Features

This kind of method depends on the presence of characters in the license plate, which results in significant change in the grey-scale level between characters' color and license plate background color. It also results in a high edge density area due to color transition.

3.1.3 License Plate Extraction Using Color Features

Since some countries have specific colors for their license plates, some reported work involves the extraction of license plates by locating their colors in the image.

The basic idea is that the color combination of a plate and characters is unique, and this combination occurs almost only in a plate region.

Ex: the specific formats of Chinese license plates

3.1.4 License Plate Extraction Using Edge Information

Since the license plate normally has a rectangular shape with a known aspect ratio, it can be extracted by finding all possible rectangles in the image. This algorithm depends on two Edge detection methods:

3.1.4.1 Sobel filter

Sobel filter is used to detect edges. Due to the color transition between the license plate and the car body, the boundary of the license plate is represented by edges in the image. The edges are two horizontal lines when performing horizontal edge detection, two vertical lines when performing vertical edge detection, and a complete rectangle when performing both at the same time.

The license plate rectangle is detected by using the geometric attribute for locating lines forming a rectangle.

Candidate regions are generated in by matching between vertical edges only. The magnitude of the vertical edges on the license plate is considered a robust extraction feature, while using the horizontal edges only can result in errors due to car bumper.

3.4.1.2 Canny filter

The Canny Edge detector was developed by John F. Canny in 1986.
Also known to many as the optimal detector, canny algorithm aims to satisfy three main criteria:

- Low error rate: Meaning a good detection of only existent edges.
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
- Minimal response: Only one detector response per edge.

As we see there is a lot of detection algorithms. After a lot of search, we reach to use edge detection algorithm in our project for extracting the plate due to the following reasons:

- The process is simple, fast and straightforward
- It has one problem as it is hardly applied to complex images since they are too sensitive to unwanted edges. but our project doesn't have complex images

3.2 Steps of detecting the plate:

3.2.1 Reading the image.

We use ***imread*** function to read the input image, and then we store the image into matrix to use it in processing.
Ex:

```
Mat image1=  imread("DSC_0122.JPG");
```

3.2.2 Convert the input image to gray

Using cvtColor function
Ex:

```
//convert to gray
cvtColor( image1, GrayImage, CV_RGB2GRAY );
```

where:
GrayImage → is the output array after conversion.
CV_RGB2GRAY → is a constant that convert from RGB to Gray image.

fig 3.1 Input Image

fig 3.2 Gray Image

3.2.3 Gaussian Blurring

Before processing the input image, it passes throw filter to reduce the noise in this image. Gaussian filter is performed to smoothing the image.

Theory: Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noises. It actually removes high frequency content from the image. So edges are blurred a little bit in this operation.

Gaussian Blurring

In this, Gaussian kernel is used. It is done with the function, *GaussianBlur().* We should specify the width and height of kernel which should be positive and odd. We also should specify the standard deviation in X and Y direction, sigmaX and sigmaY respectively. If only sigmaX is specified, sigmaY is taken as same as sigmaX. If both are given as zeros, they are calculated from kernel size. Gaussian blurring is highly effective in removing gaussian noise from the image.

Ex:

```
//gaussian filtter

GaussianBlur( GrayImage, gauss_output, Size( 3, 3 ), 0, 0 );
```

Gauss output → the output image after performing gauss filter

Size(3, 3) → the gauss kernel size is 3*3

0,0 → sigmaX and sigmaY

Fig 3.3 Gray Image

Fig 3.4 Gaussian output

3.2.4 Convert to binary

By threshold function we convert the gray image to binary image as we will complete our work on binary image. The conversion technique that is done by the threshold function follow the following algorithm. The function takes each pixel and check if the pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).

Types of threshold:

3.2.4.1 Adaptive Thresholding

The algorithm calculates the threshold for a small regions of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination. It has three 'special' input parameters and only one output argument.
Syntax:

adabtiveThreshold (InputArray **src**, OutputArray **dst**, double **maxValue**, int **adaptiveMethod**, int **threshold Type**, int **blockSize**, double **C**);

Adaptive Method → It decides how thresholding value is calculating it has two types:

1. ADAPTIVE_THRESH_MEAN_C:
   threshold value is the mean of neighborhood area.
2. ADAPTIVE_THRESH_GAUSSIAN_C: threshold value is the weighted sum of neighborhood values where weights are a Gaussian window.

Block Size → It decides the size of neighborhood area.

3.2.4.2 Otsu's Binarization

In global thresholding, we used an arbitrary value for threshold value. So, how can we know a value we selected is good or not? Answer is, trial and error method. But consider a bimodal image (In simple words, bimodal image is an image whose histogram has two peaks). For that image, we can approximately take a value in the middle of those peaks as threshold value that is what Otsu binarization does. So in simple words, it automatically calculates a threshold value from image histogram for a bimodal image. (For images which are not bimodal, binarization won't be accurate).

For this, our threshold() function is used, but pass an extra flag, cv2.THRESH_OTSU. For threshold value, simply pass zero. Then the algorithm finds the optimal threshold value and returns you as the second output, retVal. If Otsu thresholding is not used, retVal is same as the threshold value that used.

Syntax:

```
// threshold
double thresh= threshold(gauss_output, ThresholdImage, 0, 255, THRESH_BINARY|THRESH_OTSU);
```

thresh → is the returned value from threshold.
THRESH_BINARY|THRESH_OTSU → threshold type

3.2.4.3 Fixed Threshold

In this type a fixed threshold level is applied to each array element.
Syntax:

> **threshold** (InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**);

thresh → threshold value.
Maxval → maximum value to threshold.
Type → threshold type.
Threshold has five types:

1. THRESH_BINARY
2. THRESH_BINARY_INV
3. THRESH_TRUNC
4. THRESH_TOZERO
5. THRESH_TOZERO_INV

As we see we have a lot of threshold algorithms, we have to choose one of them. After a lot of attempts we choose the *Otsu's Binarization* algorithm it give us a better result than the others.
In figure 3.5 and 3.6 we introduce the result of gray image after Otsu's Binarization threshold.

3.2.5 Find edges

By using canny function, we find the vertical and horizontal edges. we choose canny filter to perform this step due to

- Low error rate: Meaning a good detection of only existent edges
- Good localization: The distance between edge pixels detected and real edge pixels have to be minimized.
- Minimal response: Only one detector response per edge.

Fig 3.5 input gray Image

Fig 3.6 Result of Otsu's Binarization threshold

Canny filter is a multi-stage algorithm and we will go through each stage.

3.2.5.1 Noise Reduction

Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with 5x5 Gaussian filter.

3.2.5.2 Finding Intensity Gradient of the Image

Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (Gx) and vertical direction (Gy). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2} \qquad \text{Eq 3.1}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \qquad \text{Eq 3.2}$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

3.2.5.3 Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient. Check the image below:

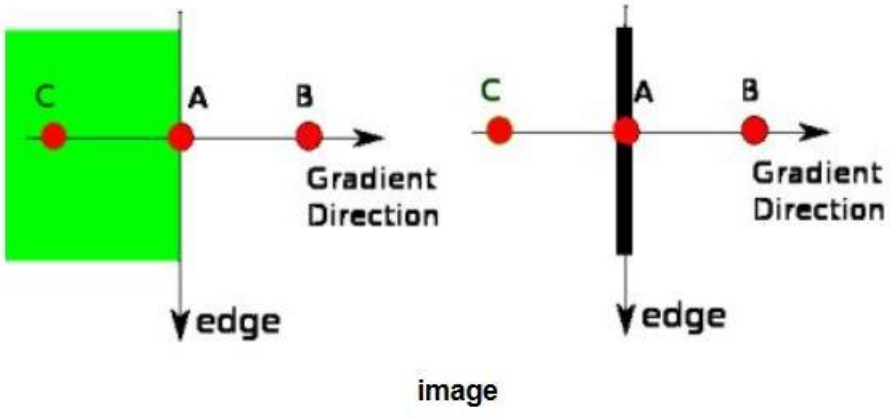

Fig 3.7

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero). *In short, the result you get is a binary image with "thin edges".*

3.2.5.4 Hysteresis Thresholding

This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the image below:

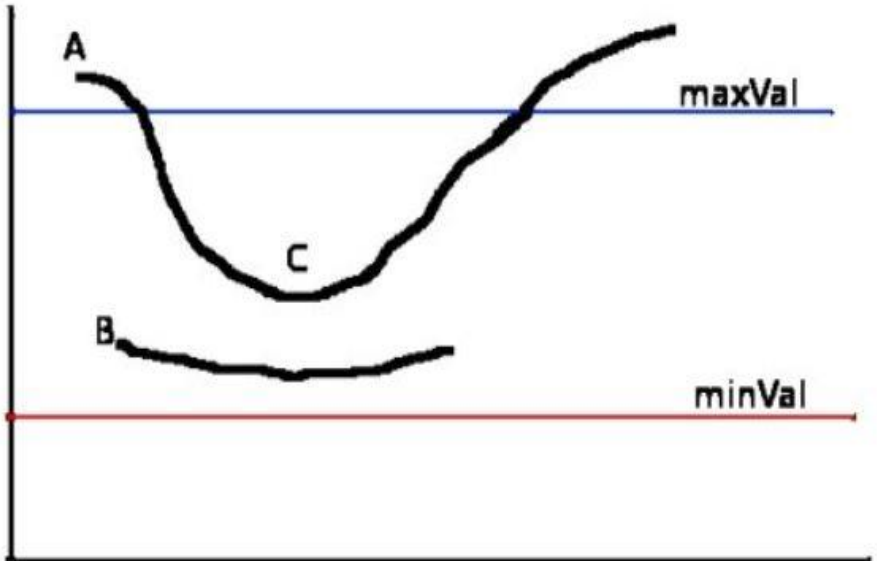

Fig 3.8

The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

This stage also removes small pixels' noises on the assumption that edges are long lines.

So what we finally get is strong edges in the image.

OpenCV puts all the above in single function, **Canny().**

```
Canny( ThresholdImage, canny, thresh1, thresh2, apertureSize, L2gradient );
```

Canny → is the output image

thresh1, thresh2 → the minimum and maximum threshold value.

after search and trying we find that the value of thresh1 should be equal to half of the threshold value that threshold function returned and the value of thesh2 should be equal to the returned value.

apertureSize → It is the size of Sobel kernel used for find image gradients. By default it is 3
L2gradient → specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned above (Eq 3.1 and 3.2) which is more accurate, otherwise it uses this function:
Edge_Gradient(G)=|G x |+|G y | Eq 3.3

Fig 3.9 Gaussian Output

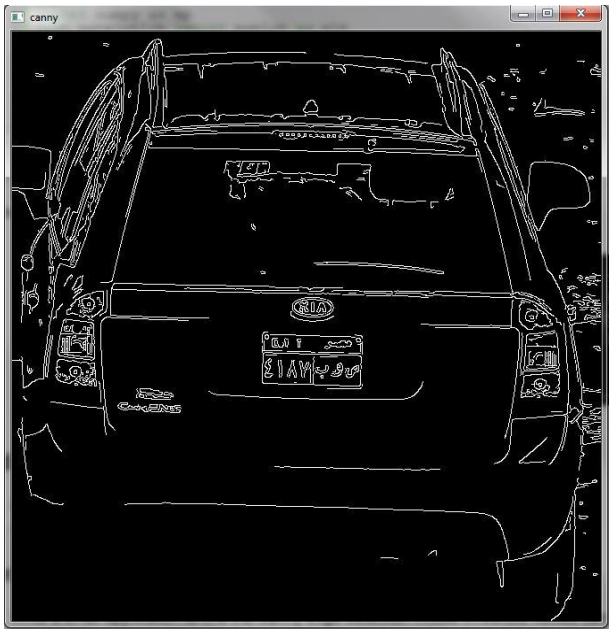

Fig 3.10 Canny Output

3.2.6 Morphological Operations

A set of operations that process images based on shapes. Morphological operations apply a structuring element to an input image and generate an output image.
The most basic morphological operations are two: *Erosion* and *Dilation*. They have a wide array of uses:

1. Removing noise
2. Isolation of individual elements and joining disparate elements in an image.
   Finding of intensity bumps or holes in an image

3.2.6.1 Erosion

The basic idea of erosion is just like soil erosion only; it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what it does? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or

simply white region decreases in the image. It is useful for removing small white noises detach two connected objects.

Fig 3.11 before erode

Fig 3.12 after erode

3.2.6.2 Dilation

It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.

Fig 3.13 before dilation

Fig 3.14 After dilation

3.2.6.3 Opening

Opening is just another name of erosion followed by dilation. It is useful in removing noise.

In the following figure, in the left side is the image before opening and in the right side is the image after opening.

Fig 2.14

3.2.6.4 Closing

Closing is reverse of Opening, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points on the object. In the following figure, the image in left side is the image before opening and in the right side is after opening.

Fig 2.15

We use closing operation in our project to close small holes inside the objects.

3.2.6.5 Structuring Element

A structuring element is a shape, used to probe or interact with a given image, with the purpose of drawing conclusions on how this shape fits or misses the shapes in the image. It is typically used in morphological operations, such as dilation, erosion, opening, and closing, as well as the hit-or-miss transform.
The following figure shows the content of the array in three cases of structure element (Rectangle, Ellipse, Cross section).

```
# Rectangular Kernel
>>> cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
>>> cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
>>> cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
array([[0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0],
       [0, 0, 1, 0, 0]], dtype=uint8)
```

Fig 3.16

Morphological operation with structure element Example:

```
Mat element1=getStructuringElement(MORPH_RECT,
Size(3,3));
morphologyEx(canny, output,MORPH_DILATE, element1);
morphologyEx(output, output, MORPH_ERODE, element1);
```

Output → is the output image after closing.
MORPH_DILATE → is a constant that choose the type of Morphological operation dilation or erosion.
Element1 → Structure Element with rectangular shape.

3.2.7 Floodfill

Fills a connected component with the given color.

```
FloodFill (InputOutputArray image, Point seedPoint,
Scalar newVal, Rect* rect=0, Scalar loDiff=Scalar(),
Scalar upDiff=Scalar(), int flags=4 );
```

seedPoint → Starting point.
newVal → New value of the repainted domain pixels.
loDiff → Maximal lower brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component.

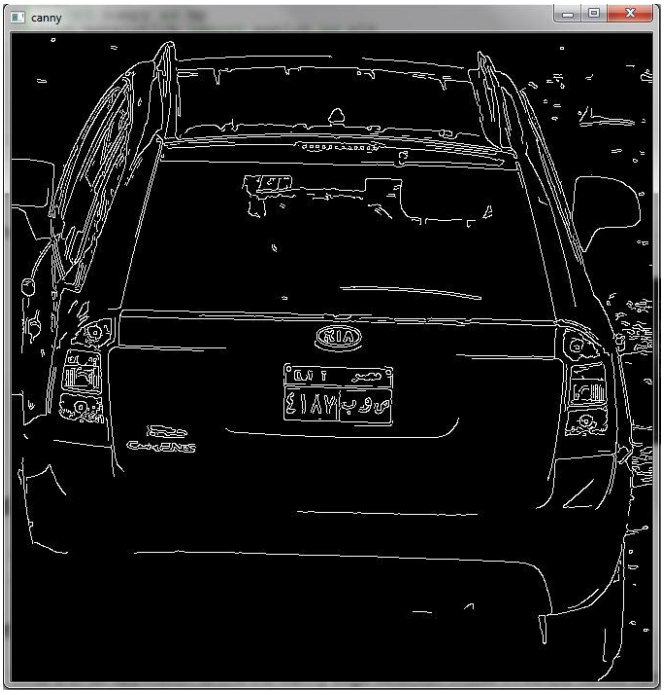

Fig 3.17 Canny output Before closing

Fig 3.18 After Closing

upDiff → Maximal upper brightness/color difference between the currently observed pixel and one of its neighbors belonging to the component, or a seed pixel being added to the component.
rect → Optional output parameter set by the function to the minimum bounding rectangle of the repainted domain.
Flags → Operation flags. The first 8 bits contain a connectivity value. The default value of 4 means that only the four nearest neighbor pixels (those that share an edge) are considered. A connectivity value of 8 means that the eight nearest neighbor pixels (those that share a corner) will be considered

Steps of filling objects:

1. Filling borders using floodfill (see figure 3.19).
2. Invert floodfilled image

```
bitwise_not(edgesNeg, edgesNeg);// inverse the floodfill_copy
```

3. Combine the two images to get the foreground

```
//or mask1 & the inverse of floodfill_copy
filledEdgesOut = (edgesNeg | mask1);
```

4. Perform closing morphological operations to fill remaining holes
Figure 3.20 and 3.21 show you the result of floodfill.

```
for (int i =0; i < mask.cols; i++) {
    if (mask.at<char>(0, i) == 0) {
        floodFill(mask,Point(i, 0), 255 , 0, 20, 20);
    }
    if (mask.at<char>(mask.rows-1, i) == 0) {
        floodFill(mask,Point(i, mask.rows-1), 255, 0,20,20);
    }
}
for (int i = 0; i < mask.rows; i++) {
    if (mask.at<char>(i, 0) == 0) {
        floodFill(mask, Point(0, i),255,0, 20, 20);
    }
    if (mask.at<char>(i, mask.cols-1) == 0) {
        floodFill(mask, Point(mask.cols-1, i),255,0, 20,20);
    }
}
```

Fig 3.19 Floodfill code

Fig 3.20 output from closing

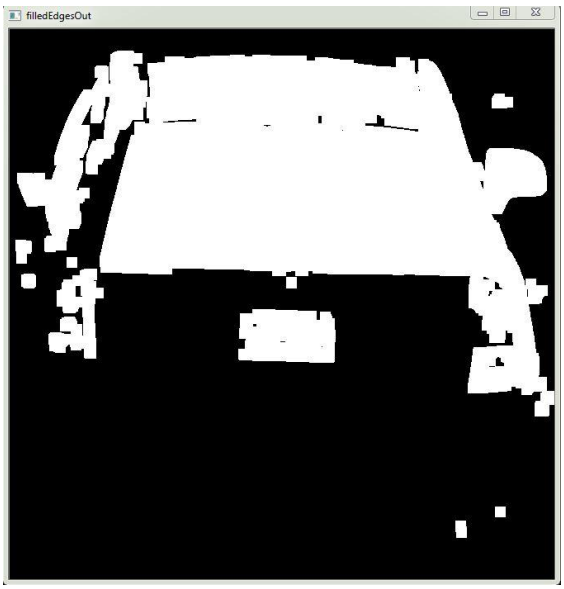

fig 3.21 After Floodfill

3.2.8 Extracting the plate

We used ***findcontour*** function to find the connected component in the image. Contours: can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition, finding contours is like finding white object from black background.

Ex:

```
findContours(InputOutputArray image, OutputArrayOfArrays
contours, OutputArray hierarchy, int mode, int method,
Point offset=Point());
```

Hierarchy → Optional output vector, containing information about the image topology. It has as many elements as the number of contours

Mode → Contour retrieval mode.

1. CV_RETR_LIST retrieves all of the contours without establishing any hierarchical relationships.
2. CV_RETR_EXTERNAL retrieves only the extreme outer contours.
3. CV_RETR_CCOMP retrieves all of the contours and organizes them into a two-level hierarchy. At the top level, there are external boundaries of the components. At the second level, there are boundaries of the holes. If there is another contour inside a hole of a connected component, it is still put at the top level.
4. CV_RETR_TREE retrieves all of the contours and reconstructs a full hierarchy of nested contours.

method → Contour approximation method.

1. CV_CHAIN_APPROX_NONE stores absolutely all the contour points. That is, any 2 subsequent points (x1,y1) and (x2,y2) of the contour will be either horizontal, vertical or diagonal neighbors
2. CV_CHAIN_APPROX_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, an up-right rectangular contour is encoded with 4 points.
3. CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS applies one of the flavors of the Teh-Chin chain approximation algorithm.

Offset → Optional offset by which every contour point is shifted. This is useful if the contours are extracted from the image ROI and then they should be analyzed in the whole image context.

Steps for extracting the plate:

1. Approximation

It approximates a contour shape to another shape with less number of vertices. Approximation made by using approxPolyDP function.

```
approxPolyDP(countor1[i], result, double epslon, bool
closed);
```

countor1 → the input contour.
result → Result of the approximation.
epslon → Parameter specifying the approximation accuracy. This is the maximum distance between the original curve and its approximation
closed → If true, the approximated curve is closed (its first and last vertices are connected). Otherwise, it is not closed.

2. Check the rectangular shape

Check if the number of corners of the contour is equal to 4 if this condition achieved the rectangular shape feature and may be a license plate.

a. Check the Aspect ratio

Calculate the width and the height of the contour and then calculate the ratio between width and height (width/height), this ratio should be between (larger than 0.85 of the width/height ratio and smaller than 1.75 of this ratio). if the contour achieved the second feature the possibility of being plate increase

b. Check area

Calculate the area of the contour and check if the area between specific values (larger than 3000 and smaller than 10000).

3. Cropping the plate

If the contour achieved the previous conditions it may be the license plate. we extract it from the gray image by using its dimensions.

First we draw a rectangle with the dimensions of the contour

```
//cropping the plate
cv::Rect myrect((int)result[indix].x,(int)result[indix].y,width,height);
```

then we use this rectangle in cropping the plate from the gray image

```
cropped = GrayImage2(myrect);
```

Fig 2.22 License Plate

3.3 Segmentation

The isolated license plate is then segmented to extract the characters for recognition.

An extracted license plate from the previous stage may have some problems, such as tilt and no uniform brightness. The segmentation algorithms should overcome all of these problems in a preprocessing step.

We categorize the existing license plate segmentation methods based on the features they used:

1. *License Plate Segmentation Using Pixel Connectivity*

Segmentation is performed by labeling the connected pixels in the binary license plate image. The labeled pixels are analyzed and those which have the same size and aspect ratio of the characters are considered as license plate characters.

Problems of this method:

Fails to extract all the characters when there are joined or broken characters.

2. *License Plate Segmentation Using Projection Profiles*

Since characters and license plate backgrounds have different colors, they have opposite binary values in the binary image.

3. *License Plate Segmentation Using Prior Knowledge of Characters*

Prior knowledge of characters can help the segmentation of the license plate. This can be performed in many ways:

a. The binary image is scanned by a horizontal line to find the starting and ending positions of the characters. When the ratio between characters' pixels to background pixels in this line exceeds a certain threshold after being lower than this threshold, this is considered as the starting position of the characters. The opposite is done to find the ending position of the characters.

b. The extracted license plate is resized into a known template size. In this template, all character positions are known. After resizing, the same positions are extracted to be the characters. This method has the advantage of simplicity.

Problem of this method:

Any shift in the extracted license plate, the extraction results in background instead of characters.

4. *License Plate Segmentation Using Character Contours*

This algorithm uses find contour function to find the connected components and rejecting contours with large and small area.

We prefer last algorithm in our segmentation as it is fast, simple and accurate.

Segmentation steps:

1. Cropping the plate with 2/3 of its height from bottom (Fig 3.23) to extract the numbers and letters region due to the characteristic of the Egyptian license plate, then convert it to binary by threshold.

Fig 3.23

2. Find contour
3. Finding minimum and maximum point in y axis to find the height of the rectangle.
4. We use the rectangle in cropping the characters (Fig 3.24).

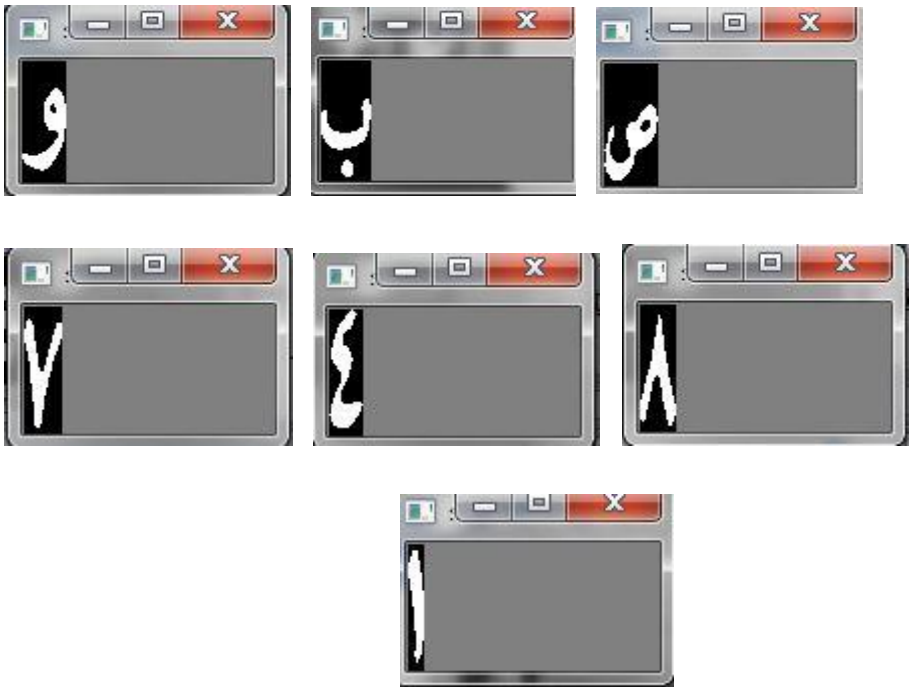

Fig 3.24 Cropped characters

Chapter 4

Arabic Optical Character Recognition

4.1 Overview

It is generally easy for a person to differentiate the sound of a human voice, from that of a violin; a handwritten numeral "3," from an "8 and the shape of a table from a chair!. However, it is difficult for a programmable computer to solve these kinds of perceptual problems. This problem _or as it's called **Pattern Recognition** _ is difficult because each object has number of characteristics that distinguish it over other related objects.

4.2 Pattern Recognition:

Pattern recognition is the science of making inferences from perceptual data, using tools from statistics, probability, computational geometry, machine learning, signal processing, and algorithm design. Thus, it is of central importance to artificial intelligence and computer vision, and has far-reaching applications in engineering, science, medicine, and business. In Figure 4.1 we will see that,

Sensor:

Which is usually a high resolution camera.

Processing Enhancement:

Using some techniques as smoothing and Gaussian Filter and morphological operations as Hysteresis Threshold, Or Resize the image to a convenient value (IN our test case it was 40 * 75 Pixels).

Feature Extraction:

Which means extracting the features of an object in a vector rather than dealing with the whole image each time. In our test case we explored 3 Algorithms PCA, Hu-Moment and Projection profile.

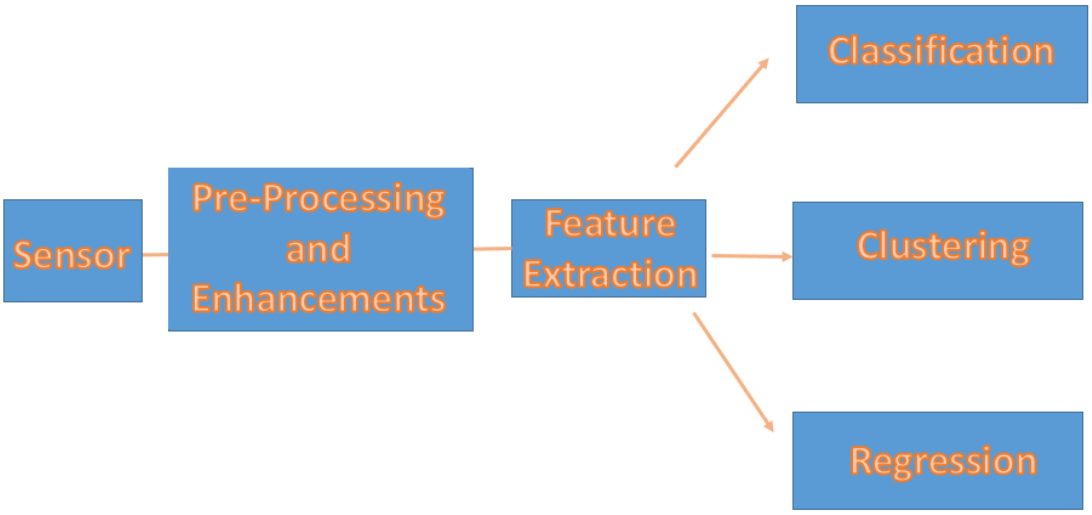

Fig 4. 1 : The Process of Pattern recognition

Classification:

- In general, classification process is when you have a set of predefined classes and want to know which class a new object belongs to.
- The classification based on the availabilityof a set of patterns that have already been classified or described.
- This set of patterns is termed the **training Set** This learning strategy is characterized as **supervised Learning.**

Clustering:

The system is not given any priori labelling of patterns, instead it establishes the lasses itself based on the statistical regularities of the patterns.

4.3 Optical character recognition

Optical character recognition refers to the branch of computer science that involves reading text from paper and translating the images into a form that the computer can manipulate (for example, into ASCII codes). An OCR system enables you to take a book or a magazine article, feed it directly into an electronic computer file, and then edit the file using a word processor.

In our system OCR used in license plate recognition which involves capturing images of license plate. After segmentation step the characters and numbers were cut into blocks with fixed size of 40*75. They are processed by series of algorithms that are able to match segmented blocks with previous stored templates of 17 characters and 9 numerical.

4.3.1 Recognition Process

Recognition process is divided into two steps Feature Extraction and classification. First, let us discuss the feature Extraction Algorithms.

4.3.2 Feature Extraction

Feature is “a distinctive attribute or aspect of something." so the thing is to have some set of values for a particular instance that diverse that instance from the counterparts. In the field of images, features might be raw pixels for simple problems like digit recognition of well-known dataset. However, in natural images, usage of simple image pixels is not descriptive enough. Instead there are two main steam to follow. One is to use hand engineered feature extraction methods (e.g. SIFT, VLAD, HOG, GIST, LBP) and the another stream is to learn features that are discriminative in the given context (i.e. Sparse Coding, Auto Encoders, Restricted Boltzmann Machines, PCA, ICA, K-means). Note that second alternative, representation learning, is the hot wheeled way nowadays.

4.3.3 Principal Component Algorithm

Principal component analysis is probably the oldest and best known of the techniques of multivariate analysis. The PCA was discovered by Pearson in 1901 and then independently developed by Hoteling in 1933, by Karhunen in 1947 and by Lomeve.

PCA is a procedure of finding the so called principal components of observed data presented by a large random vector, i.e. of components of a smaller vector which preserves principal features of observed data. In particular, this means that the original vector can be reconstructed from the smaller one with the least possible error.

Fig 4. 2 Original Image

Fig 4. 3 Reconstructed Image

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. PCA can be done by eigenvalue decomposition of a data covariance or singular value decomposition of a data matrix, usually after mean centering and normalizing the data matrix for each attribute.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualized as a set of coordinates in a high-dimensional data space (1 axis per variable), PCA can supply the user with a lower-dimensional picture, a projection or "shadow" of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced.

4.3.3.1 PCA mathematical theory

PCA is mathematically defined as an orthogonal linear transformation that transform the data to a new coordinate system such that the greatest variance by some projection of the data comes to lie on the first coordinate (called the first principal component).

Computing PCA using the covariance method

To calculate PCA at the first collect some of data or samples to the object you need to recognize.

Suppose you have data comprising a set of observations of p variables, and you want to reduce the data so that each observation can be described with only L variables, $L < p$. Suppose further, that the data are arranged as a set of n data vectorsX1…….Xn with each Xi representing a single grouped observation of the p variables.

- Write X1……..Xn as row vectors, each of which has p columns.
- Place the row vectors into a single matrix **X** of dimensions $n \times p$.

Calculate the empirical mean

$$m(i,j) = \frac{1}{n}\sum_{i=1}^{n} X(i,j)$$

Calculate the deviations from the mean

- Subtract the empirical mean vector **m** from each row of the data matrix **X**.
- Store mean-subtracted data in the $n \times p$ matrix **B**

$$B = X - hm^T$$

Where **h** is an $n \times 1$ column vector of all 1s.

$$h[i] = 1 \quad for\ i = 1, \dots\dots\dots\dots\dots, n$$

Find the covariance matrix

Find the p × p empirical covariance matrix C from the outer product of matrix B with itself.

$$C = \frac{1}{n-1} B^* . B$$

Where B^* is the conjugate transpose operator? Note that if **B** consists entirely of real numbers, which is the case in many applications, the "conjugate transpose" is the same as the regular transpose.

Find the eigenvectors and eigenvalues of the covariance matrix

Compute the matrix **V** of eigenvectors which diagonalizes the covariance matrix **C**.

$$V^{-1}CV = D$$

Where **D** is the diagonal matrix of eigenvalues of **C.**

Matrix **V**, also of dimension $p \times p$, contains p column vectors, each of length p, which represent the p eigenvectors of the covariance matrix **C**.

Rearrange the eigenvectors and eigenvalues

Sort the columns of the eigenvector matrix **V** and eigenvalue matrix **D** in order of *decreasing* eigenvalue.

Make sure to maintain the correct pairings between the columns in each matrix.

Compute the feature vector of input image

- Convert the image Mat to vector Vi
- Calculate the feature vector of Vi by equation

$$Vf = V * (Vi - m)$$

Examples on the PCA:

Here's some illustrative Examples about How the PCA works.

1. The Re-construction of the Source image using the feature vector.

Fig 4. 4 Original Image

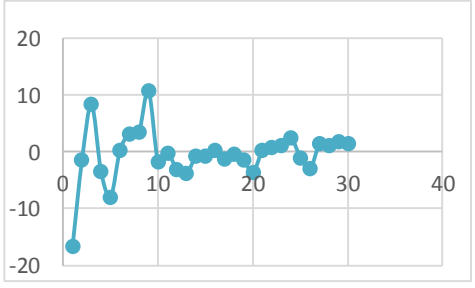

Fig 4. 5 Feature Vector

Fig 4. 6 Reconstructed img

Fig 4. 7 Original Image

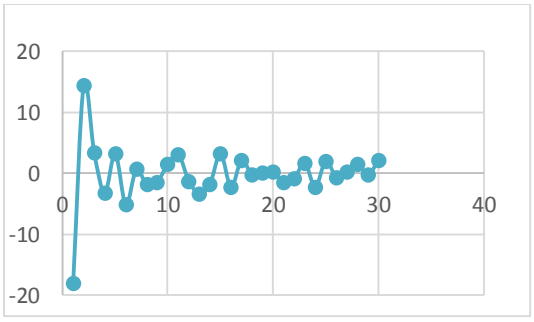

Fig 4. 8 Feature Vector

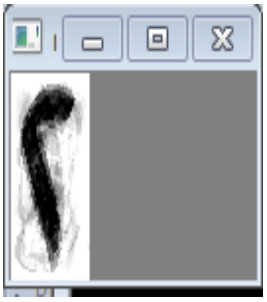

Fig 4.9 Reconstructed image

2. The feature vector constructed using the PCA.

☐ PCA Feature vector for character (2):

Sample Image 1	Sample Image 2	Sample Image 3
 *Fig 4.10*	 *Fig 4. 11*	 *Fig 4.12*

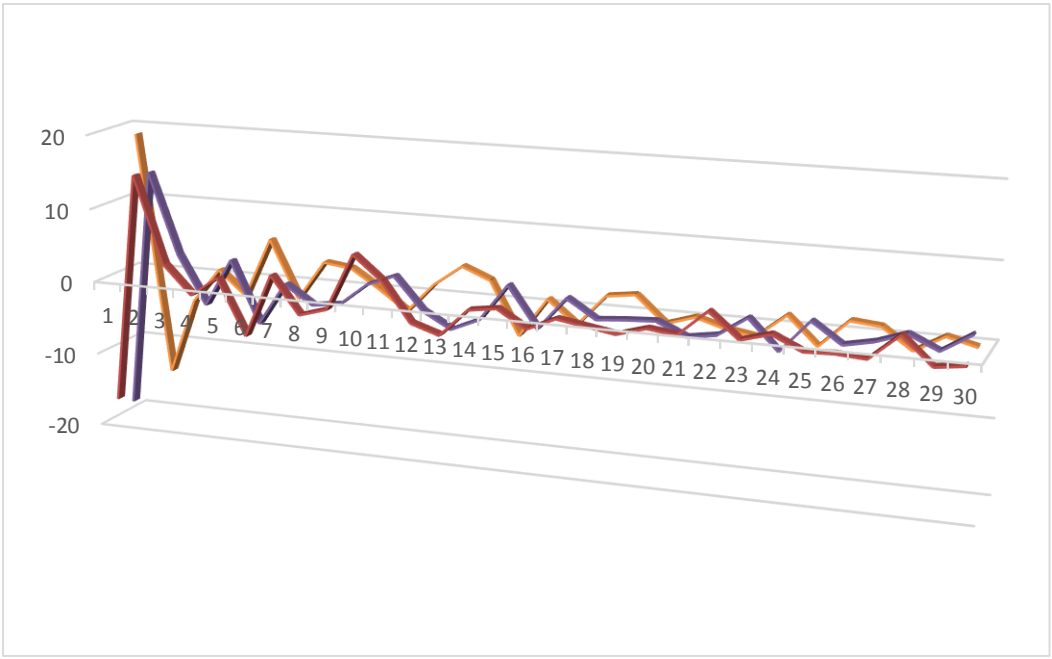

Fig 4. 13
PCA Feature vector of three samples

- PCA Feature vector for character (ب):

Sample Image 1	Sample Image 2	Sample Image 3
		
Fig 4.14	*Fig 4. 15*	*Fig 4.16*

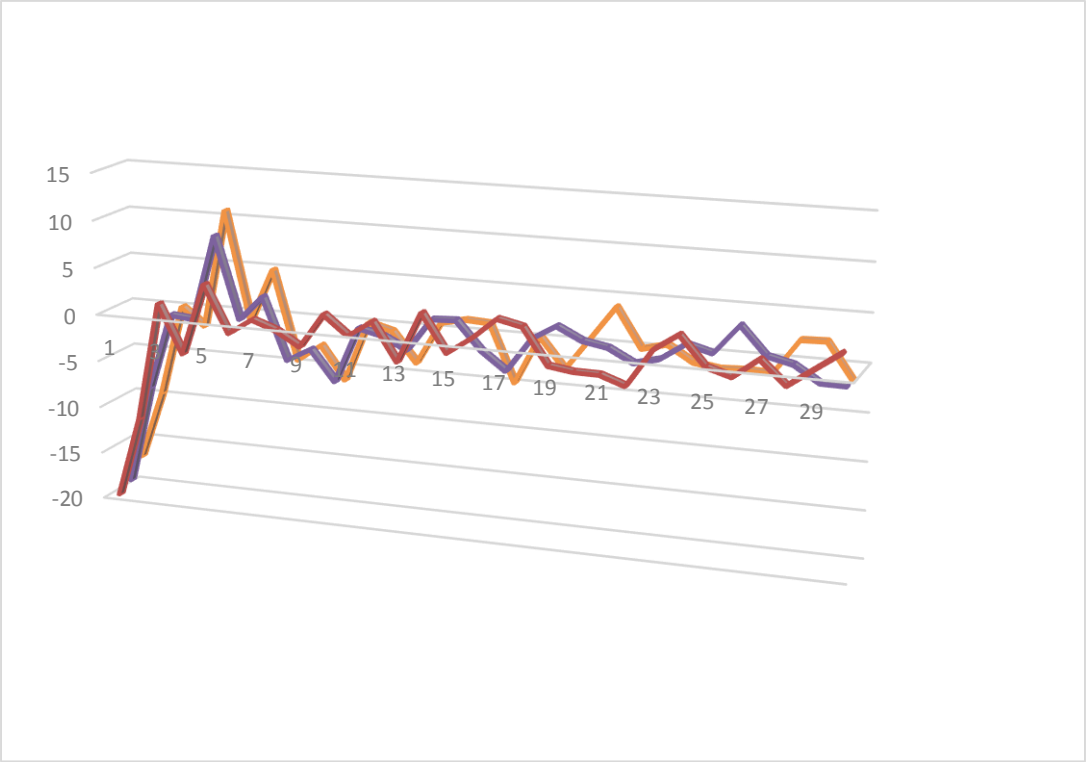

Fig 4.17
PCA Feature vector of three sample

- PCA Feature vector for character (و):

Sample Image 1	Sample Image 2	Sample Image 3
		
Fig 4.18	*Fig 4. 19*	*Fig 4.20*

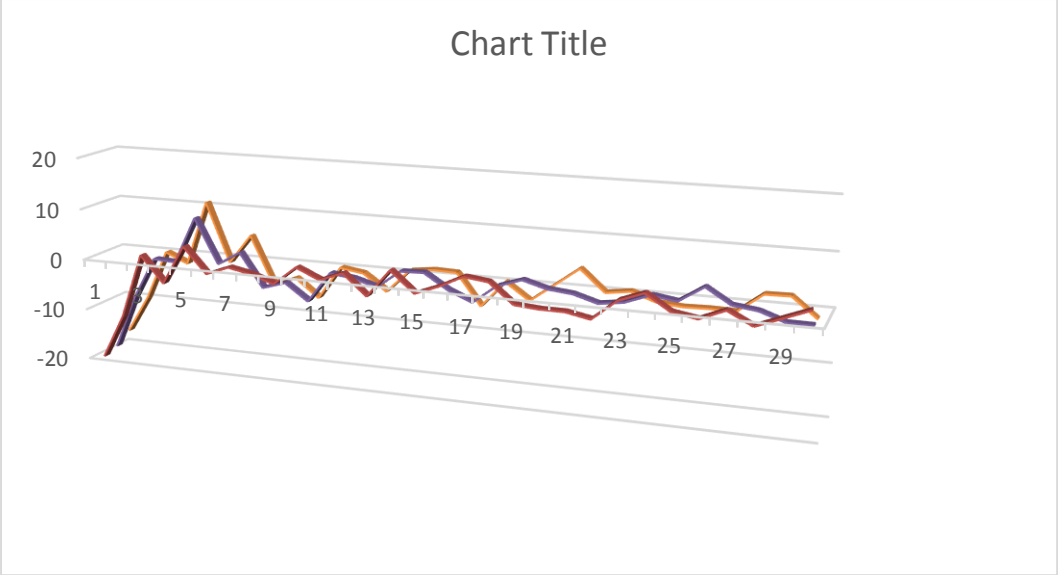

Fig 4. 21
PCA Feature vector of three samples

It's obvious from the previous figures that the Sample Images are almost similar. And That each feature vector from one Character is different from the other characters feature vectors.

That's what we will see with the other Feature extraction Algorithms as well.

4.3.3.2 **Hu Moment Algorithm**

Moment invariants used as features for image processing, shape recognition and classification. Moments can provide characteristics of an object that uniquely represent its shape.

HU Moments invariant values are invariant with respect to translation, scale and rotation of the shape.Hu Moment invariants guarantee to be constant for given shape no matter how it's shifted. Rotated or scaled.

From the second and third order values of the normalized central moments a set of seven invariant moments can be computed.

Hu Moment mathematical theory

Moment invariants are computed based on the information provided by both the shape boundary and its interior region.

The regular moments are defined by:

$$M_{ji} = \sum_{x,y} (array(x,y).x^j.y^i)$$

The central moment computed as:

$$mu_{ji} = \sum_{x,y} (array(x,y).(x-\bar{x})^j.(y-\bar{y})^i)$$

Where ($\bar{x}$, $\bar{y}$) are center mass:

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad , \bar{y} = \frac{m_{01}}{m_{00}}$$

The normalized central moments computed as

$$nu_{ji} = \frac{mu_{ji}}{m00^{(i+j)/2+1}}$$

Hu Moment function

```
void HuMomonts(const Moments& m, Output Array hu) .
```

Parameters:

- Moments – Input moments computed with moments ().
- **hu** – Output Hu invariants .

Hu Moment function computes the seven Hu-moments from normalized central moment.

The function calculates seven Hu invariants defined as:

$$hu[0] = \eta_{20} + \eta_{02} .$$

$$hu[1] = (\eta_{20} + \eta_{02})^2 + 4\eta_{11}^2 .$$

$$hu[2] = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 .$$

$$hu[3] = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 .$$

$$hu[4] = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})(\eta_{21} +)[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

$$hu[5] = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}).$$

$$hu[6] = (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30})^2 - (\eta_{21} + \eta_{03})^2].$$

Where:

- η_{ji} Stand normalized central moments.
- m_{00} Is a mass of the image (or binary images, m_{00} is an area of the object).
- $\frac{m_{10}}{m_{00}}$ and $\frac{m_{01}}{m_{00}}$ define the centroid of the image
- Higher order moments describe the "distribution of mass" of the image.

First six Hu moment values invariant to translation rotation and scaling. Seventh one skew invariant which help distinguish mirror image.

Examples on the Hu-Moment:

In the Following Figures we present the feature vectors made using the Hu-Moment Algorithm, for 3 arbitrary chosen characters each has 3 different samples with the same size 40*75 pixels.

.

- Hu Moment for sample of Number (2)

Sample Image 1	Sample Image 2	Sample Image 3
		
Fig 4.22	*Fig 4. 23*	*Fig 4.24*

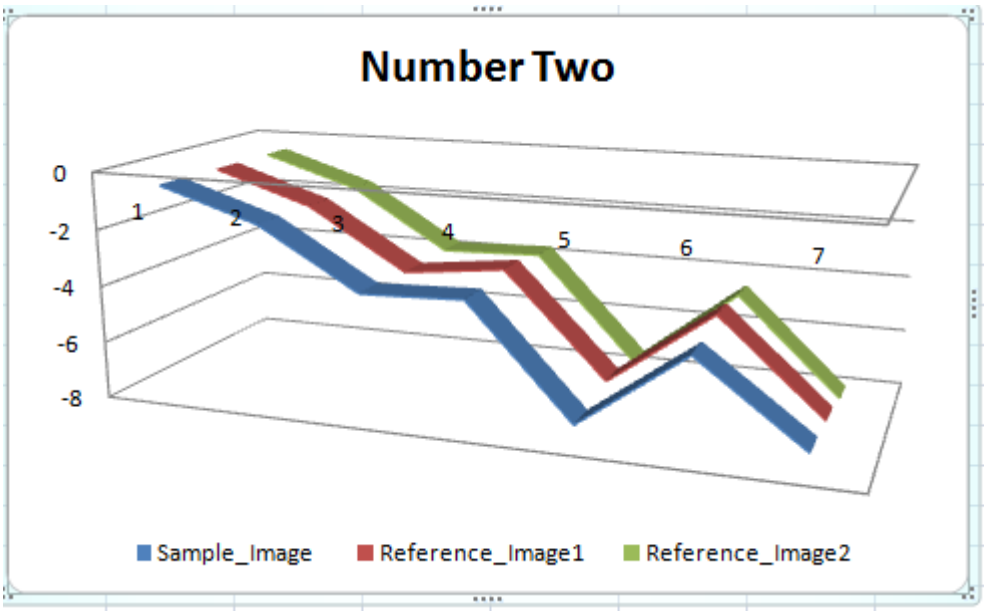

Fig 4. 25
HU Moment vector of three Samples

➢ Hu Moment for character (ب):

Sample Image 1	Sample Image 2	Sample Image 3
ب	ب	ب
Fig 4.26	*Fig 4. 27*	*Fig 4.28*

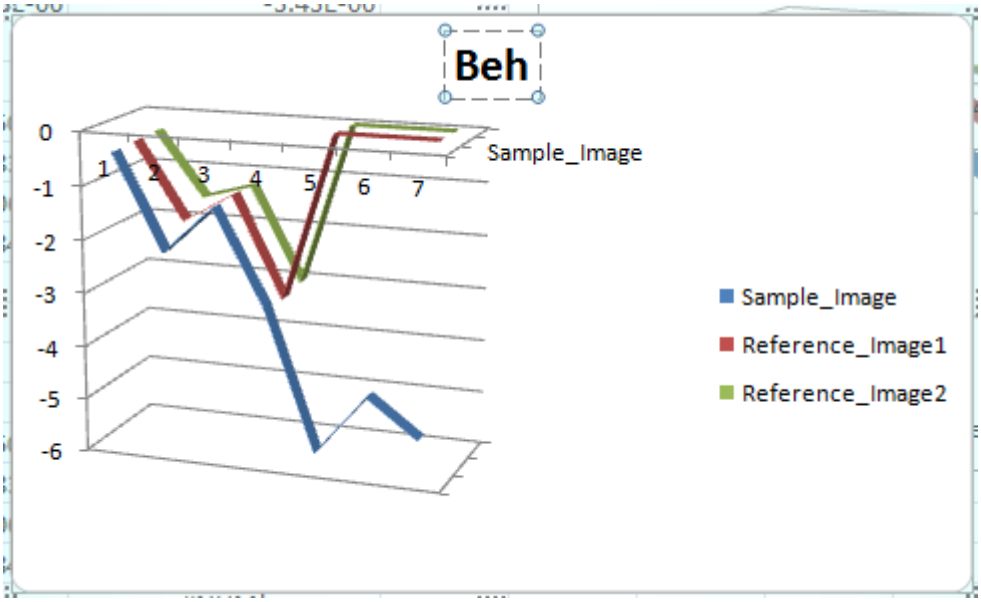

Fig 4. 29
Hu Moment of Three samples

➢ Hu Moment for character (و):

Sample Image 1	Sample Image 2	Sample Image 3
و	و	و
Fig 4.30	*Fig 4. 31*	*Fig 4.32*

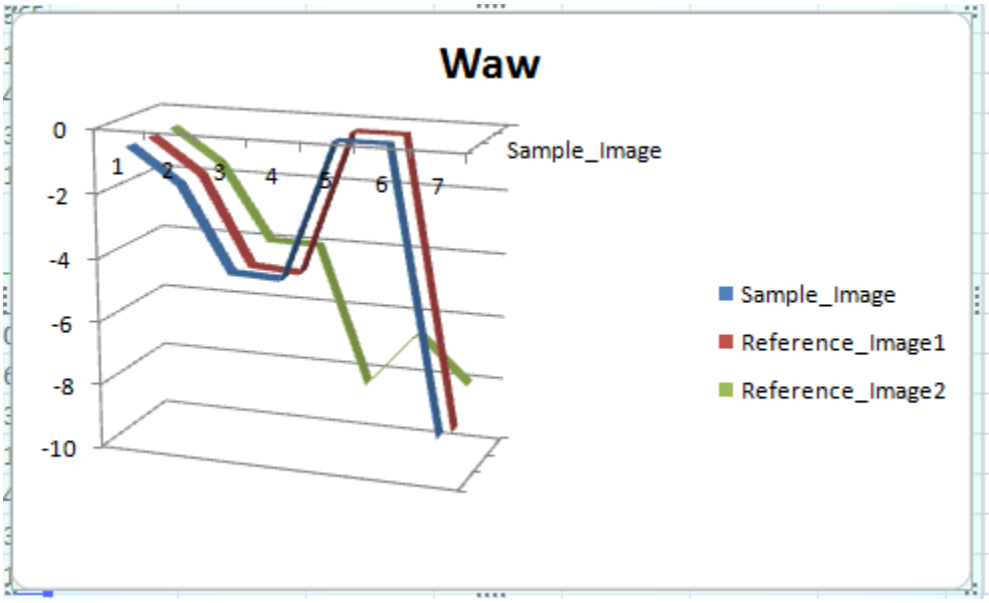

Fig 4. 33
Hu Moment of Three Samples

From the previous figures we notice that the vectors are somehow similar to each other but with some divergence among them. This divergence is why the experimental results of the Hu-Moment were not very good. You'll find all the Experimental results in Chapter 5 of this book.

4.3.3.3 Projection Profile Algorithm

In this method, the horizontal or vertical projection profile is used as a suitable feature for feature Extraction. Horizontal and / or vertical projection profile is the histogram of a one-dimensional array with a number of entries equal to the number of rows (or columns or both).

It's simple as one might think its results would be with great deal of Error …. In fact, projection profile results are quite good and reliable. You could even get better results when taking large Number of training set.

The disadvantage of the projection profile is that the feature vector has large number of feature element (No. of Row or No. of Columns or both) which results in large memory size.

In our Project we extracted the feature vector as follows:

- First, get the Horizontal projection.
- Second, get the vertical projection.
- Finally, combine both together in a larger vector.
  Note:
  Horizontal Projection only works on binary image (0's and 1's) so before we perform these steps we need to convert it to a binary image .

The function that dose that is:

```
void getFeatureVectorUsingHorizontal_VerticalProjection ( )
```

You'll find it available in appendix A.

Examples on the Projection Profile:

In the Following Figures we present the projection profile for 3 arbitrary chosen characters each has 3 different samples with the same size 40*75 pixels.

➢ Projection Profile for character (2):

Sample Image 1	Sample Image 2	Sample Image 3
 *Fig 4.34*	 *Fig 4. 35*	 *Fig 4.36*

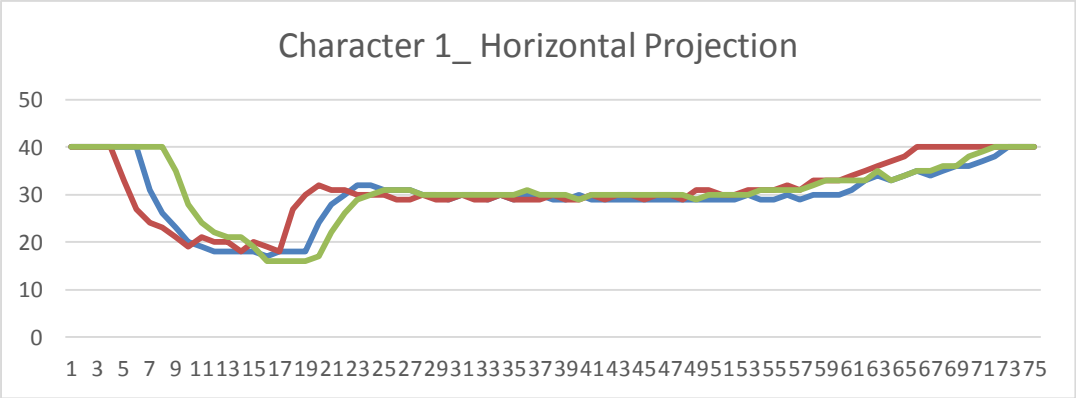

Fig 4. 37
Horizontal Projection feature vector of Three Samples

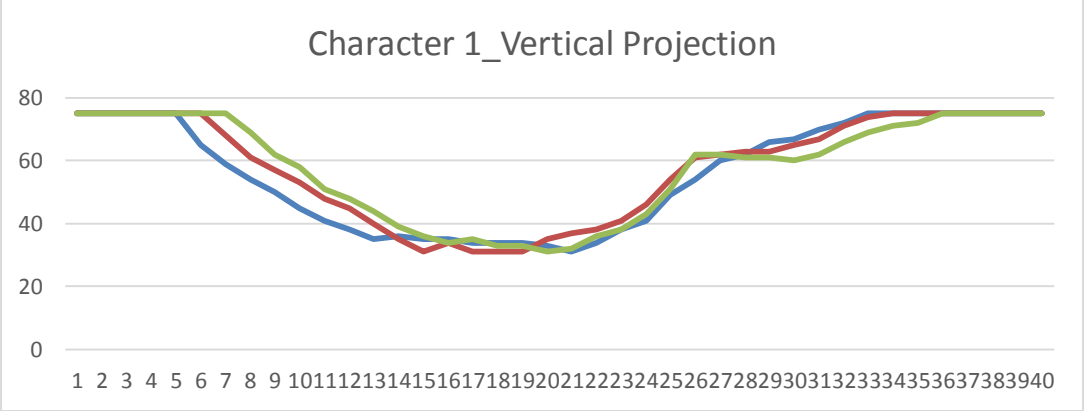

Fig 4. 38
Vertical Projection feature vector of Three Samples

➢ Projection Profile for character (ب):

Sample Image 1	Sample Image 2	Sample Image 3
ب	ب	ب
Fig 4.39	*Fig 4. 40*	*Fig 4.41*

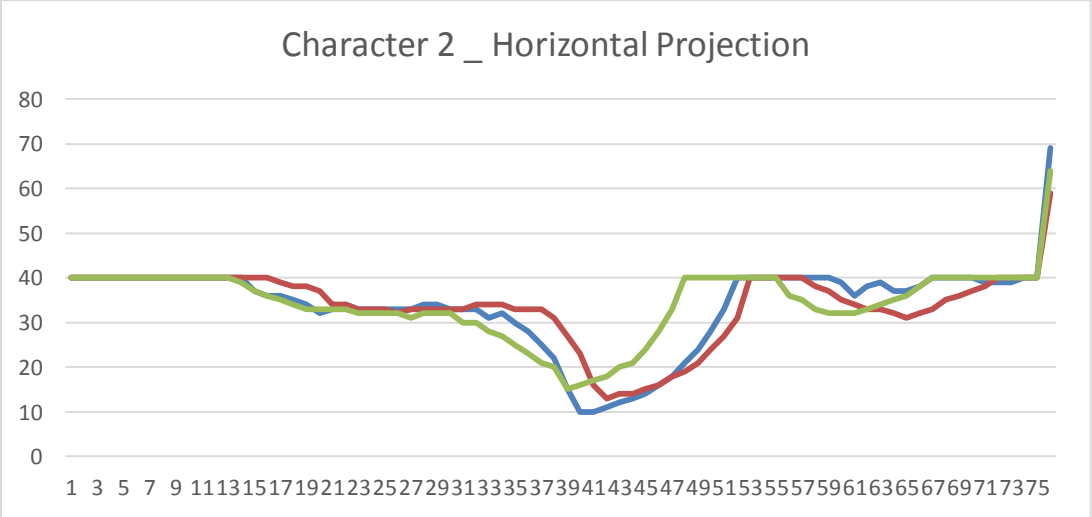

Fig 4. 42
Horizontal Projection feature vector of Three Samples

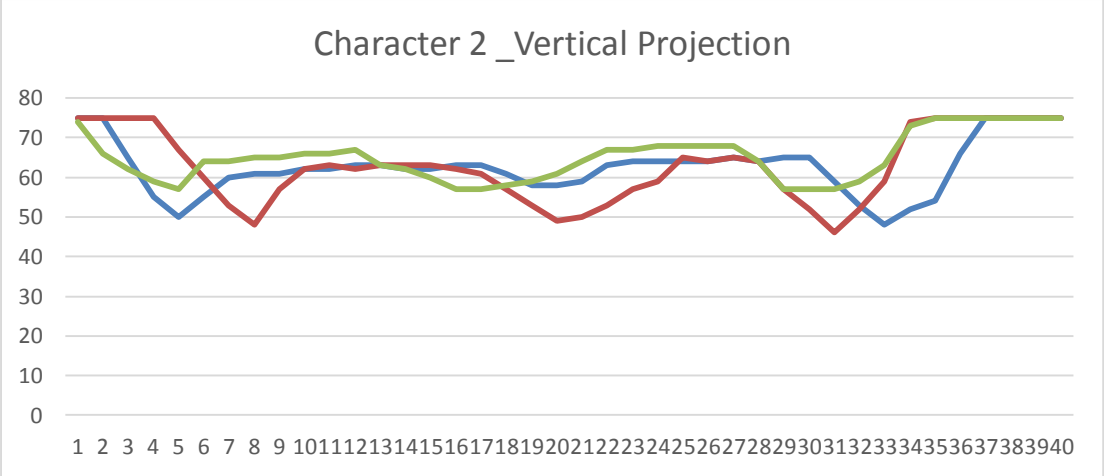

Fig 4. 43
Vertical Projection feature vector of Three Samples

Sample Image 1	Sample Image 2	Sample Image 3
		
Fig 4.44	*Fig 4. 45*	*Fig 4.46*

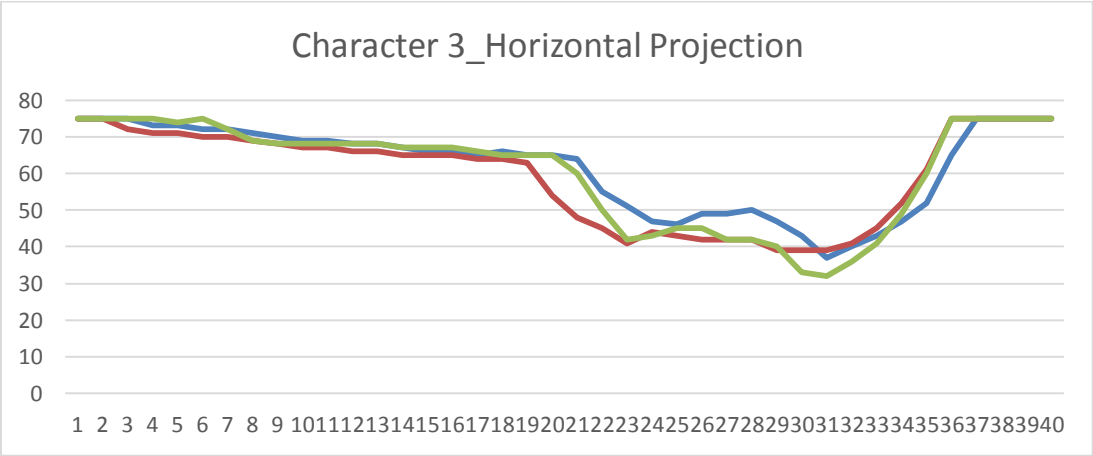

Fig 4. 47
Horizontal Projection feature vector of Three Samples

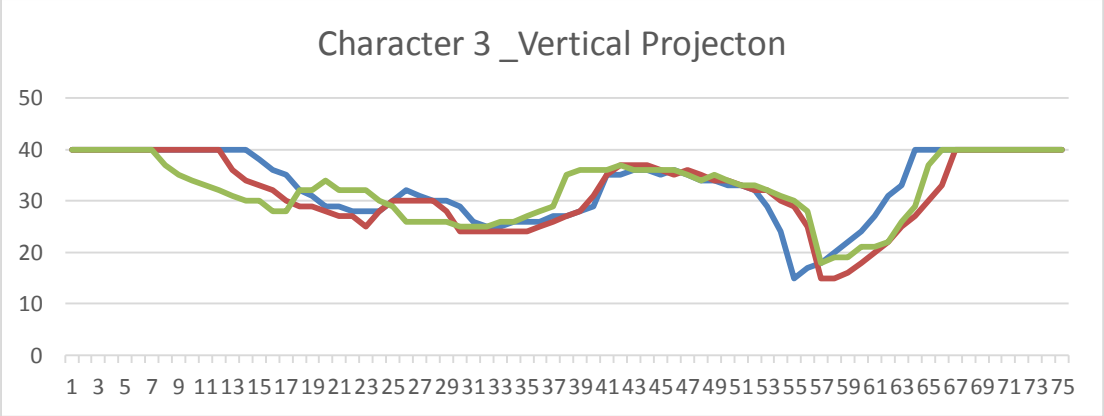

Fig 4. 48
Vertical Projection feature vector of Three Samples

We could clearly Notice that all the charts has features describing that specific character as each sample of the same character is all most the same .
Please Note that you'll get even better results if you take care of the pre-processing stage.

4.3.4 Classification:

AS We previously illustrated classification is the process of giving an unlabeled sample a label based on its similarity to other labeled samples called training set.

4.3.4.1 K Nearest Neighbors Classification

K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique.

KNN Theory

We have the Training set Represented by their feature vectors.
We extract the feature vector of the new unlabeled data for testing. Our aim is to find the class label for the new point.

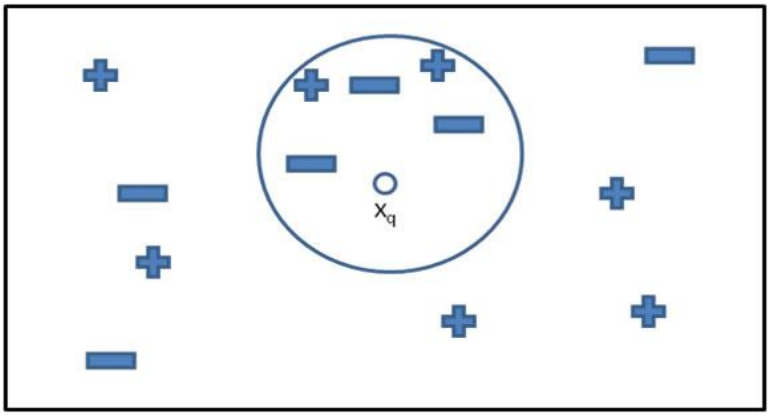

Figure 4.49
Represent the 5 nearest Neighbors for the unlabeled sample Xq

KNN Algorithm:

The KNN Algorithm is Quiet easy , it's even implemented in the opencv library , but for the sake of simplicity we used our own implemented function for KNN .

First, Prepare the training set. This is done by extracting the feature vector for each image and label it.
Second, we extract the feature vector for the unlabeled image.
Then, we calculate the similarity (Represented by the distance in our Case) between the unlabeled vector and each vector of the labeled training set vectors.
Finally, we get the labels for the K most similar vectors (least distance) and get the vote for the highest label.

Of course it might get to your mind "**what if the result is a tie?**" That's why it's important to set a suitable value for k based on your test results. Following we present some cases that you should keep in mind when selecting a value for K.

The algorithm has different behavior based on k as follows: -

Case 1: k = 1 or Nearest Neighbor Rule This will get the most similar characters to the test sample, The Nearest neighbor to the sample unlabeled image.

Case 2: k > 1

Then the problem is "which one is our sample?" Here we take the vote to get the largest Number of characters with same label within the k nearest neighbors.
What if we had a tie?! Meaning that each Neighbors has k/2 votes !!
This could lead to un-stability of the algorithm as the decisions in these cases are not the same foe all cases.

To prevent this from happening we select k to be an odd number.
The Experimental tests here were made with k=3.

4.3.4.2 Similarity Metrics:

An important Question now is "How to say **if a** character is **somehow** similar to a character more that it is to another one?
This would be done using the similarity **Measures.**

The measure reflects the degree of closeness or **Separation** of the target objects.

Is the simplest scenario. Let A be the character to be labeled. Find the character closest to A. Let it be B. Now nearest neighbor rule asks to assign the label of A to B. This seems too simple.

Note that this will probably result a huge error but reasoning holds only when the number of data points is not very large.

There is number of methods used to measure the similarity between objects (represented by their feature vector) like (Euclidian distance, cosine theta)

4.3.4.2.1 Euclidian Similarity

Euclidean distance has nothing to do with machine learning specifically. It is only one of the many available options to measure the distance between two vectors / data objects. However, many classification algorithms (for e.g. K-Nearest Neighbor, Minimum Distance Classifier etc.) use it to either train the classifier or decide the class membership of a test observation

Mathematical Theory:

The **Euclidean distance** between point's **p** and **q** is the length of the line segment connecting them ().

if $\mathbf{p} = (p_1, p_2,..., p_n)$ and $\mathbf{q} = (q_1, q_2,..., q_n)$ are two points in Euclidean *n*-space, then the distance (d) from **p** to **q**, or from **q** to **p** is given by the formula :

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

IN our case we built our own Euclidean distance function as well as the cosine theta Function both you'll find in the Appendix A.

```
double compare_CosineTheta ()

double compare_Eclidean ()
```

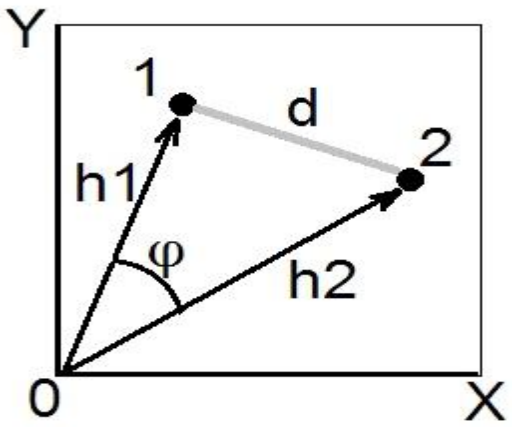

Figure 4.50
Euclidean Distance between two vectors h1 and h2

4.3.4.2.2 Cosine Similarity

Cosine similarity is a measure of similarity between two vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. It is thus a judgment of orientation and not magnitude.

Two vectors with the same orientation have a cosine similarity of 1.

Two vectors at 90° have a similarity of 0, neatly bounded in [0, 1].

Mathematical Theory:

The cosine of two vectors can be derived by using dot product formula

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta$$

Hence cosine similarity *cos (θ) is:-*

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

We could notice that the result here is an angular similarity if we wanted to convert it to distance we use the formula:

$$\text{distance} = \frac{\cos^{-1}(\text{similarity})}{\pi}$$

$$\text{similarity} = 1 - \text{distance}$$

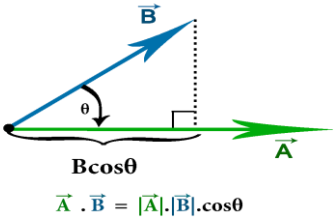

Figure 4.51
Dot Product of two vectors A and B .
Where Ai and Bi are components of vectors A and B respectively.

Example of cosine similarity:

For Two vectors it's said if they are similar or unrelated if the angle between them is small or large respectively. This is illustrated in the following figure.

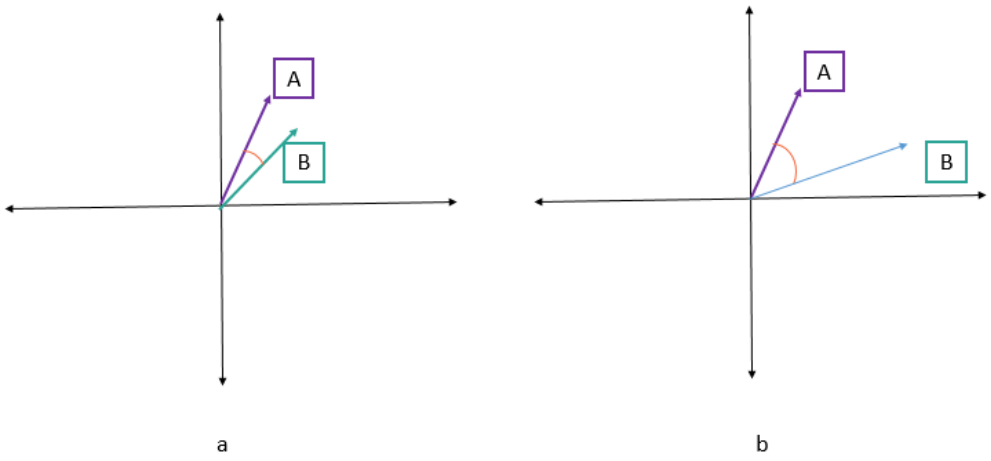

Figure 4. 52
Performing the cosine Similarity test on two vectors vector A and B
a – Represents similar scores.
b- Represents unrelated scores.

In our test case we chose to work with the Euclidean distance metric when we built the KNN function. Of course you might try to build your own KNN Using the cosine theta function.

Chapter 5

Experimental Results and Conclusions

5.1 Overview

Finally, after a hard work over this year we finish our project and get a satisfied output to each of the previous stages, in this chapter we will show you some results and statistics about each stage output.

5.2 Experiments Results

5.2.1 Detection Stage

As we know we use the edge detection algorithm in this stage. This algorithm is sensitive to any change in image (illumination, dirty plates, bad resolution, Crabs on plate, etc.)

First we will show you some pictures that has changes.

1. First example this car has a shadow on its plate and a lot of details

Fig 5.1 canny output

Fig 5.2 original image

2. Second example this plate has some crabs on it.

Fig 5.3 canny output

Fig 5.4 original image

3. Third example the image has bad resolution

Fig 5.5 canny output

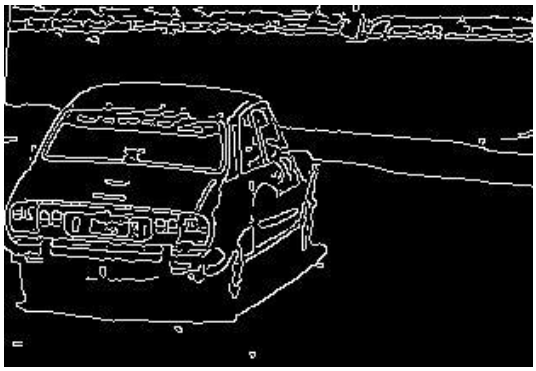

Fig 5.6 original image

The following figures are successful figures

Fig 5.7 Input image

Fig 5.8 Cropped plate

Fig 5.9 Input image

Fig 5.10 Cropped plate

Fig 5.11 Input image

Fig 5.12 Cropped plate

5.2.2 Segmentation Stage

As we know we use the find contour algorithm this algorithm finds all the contours in the plate, there may be some areas represent noises, as the noisy area may have the same area like characters so this area will be segmented by the algorithm and satisfy the area condition.

But there was a lot of successful cases and the segmentation algorithm works well. Here are some examples:

Fig 5.11 Cropped plate

Fig 5.12 Segmented characters

Fig 5.13 Cropped plate

Fig 5.14 Segmented characters

Fig 5.15 Cropped plate

Fig 5.16 Segmented characters

5.2.3 Recognition Stage

Regarding the Recognition we explored 3 Algorithms (PCA – Hu-Moment – Projection Profile) . Now we document the results for each one:
We tested each one over 21 test Samples (some of them are shown below) and we trained each Algorithm over 95 samples , 5 for each character Hence 19 characters of both Numbers and Letters :

Figure 5.17 The characters combination taken in Experimental test.

5.2.3.1 Projection profile Algorithm

The results with Projection profile were not that good. as it only recognized 60% of the samples, four out of ten test samples.

The results could be improved by improving the Pre-processing stage but we didn't try any of these techniques as the PCA presented a very High Accuracy even with the most difficult samples.
By examining table 5.1 we notice that the wrong results comes from the similarity of two or more characters in the Arabic language itself. This is illustrated in the coming Example.

Consider these two letters, a Human eye could distinguish between both unlike computers which works according to an Algorithm specified by the developer. For a computer working with the projection profile algorithm the two characters are somehow similar, which is illustrated by Figure 5.19.

Fig 5.18

Sample image	1st Neighbor	2nd Neighbor	3rd Neighbor
	Eight	Eight	Sad
	Eight	Beh	Beh
	Reh	Dal	Eight
	Waw	Reh	Waw
	Four	Four	Four
	Gem	Five	One
	Two	Two	Two
	Three	Three	Two
	Beh	Eight	Beh
	Seven	Seven	Seven

Table 5.1 results of profile projection

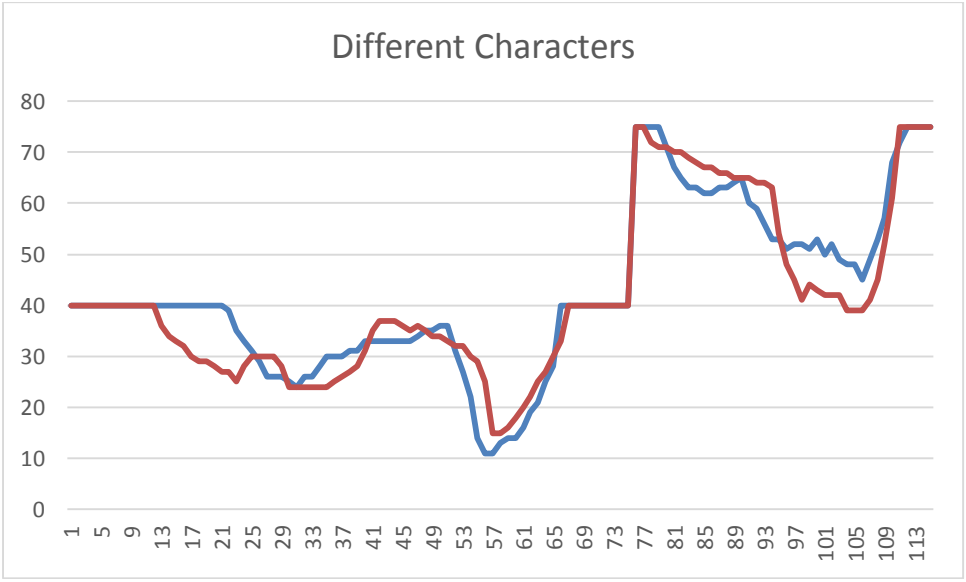

Fig 5.19 profile projection of similar samples

You'll find that they are all most similar as if they were the same character!
This is where the ambiguity comes from.

5.2.3.2 Hu-Moment Algorithm

The results with Hu-moment algorithm were only 20% - two out of ten test samples which is very bad results .

Sample image	1st Neighbor	2nd Neighbor	3rd Neighbor
	Sad	Waw	Waw
	Reh	Beh	Six
	Qaf	Waw	Sad
	Sad	Seven	Waw
	Eight	Four	Dal
	One	Five	Eight
	Five	Two	Eight
	Five	Five	Three
	Beh	Six	Beh
	Seven	Sad	Sad

Table 5.2

To get a good result from Hu Moments function must sample and reference images capture from the same camera to get the same resolution.

Hu Moments function results not good enough because Hu Moments has some drawbacks as:

1. Hu Moments produce higher error rate compared to other method (PCA or Vertical/Horizontal projection).
2. Any small local disturbance affects all moments.

3. Hu Moments are invariant to translation, rotation and scaling, but are not invariant to change in illumination.

5.2.3.3 PCA Algorithm

The results with PCA algorithm were Excellent. it nearly recognized every character we faced except for some cases where even the human eye couldn't recognize what the character is.

Sample image	1st Neighbor	2nd Neighbor	3rd Neighbor
	**Sad**	**Sad**	**Sad**
	**Tah**	**Tah**	**Reh**
	**Dal**	**Five**	**Dal**
	**Waw**	**Waw**	**Dal**
	**Four**	**Four**	**Four**
	**Five**	**Five**	**One**
	**Two**	**Two**	**Two**
	**Three**	**Three**	**Three**
	**Beh**	**Beh**	**Five**
	**Seven**	**Seven**	**Seven**

Table 5.3 PCA Results

From experimental result PCA is high sufficient algorithm in recognition.
In data of high dimensions, where graphical representation is difficult, PCA is a powerful tool for analyzing data and finding patterns in it.performance is good when noise is present.

PCA draw back

- PCA is black box that is widely used but poorly understood.
- Finding the eigenvectors and eigenvalues are time consuming.
- Performance is very bad if scale of image is changed.

Conclusion:

Comparing all Algorithms Experimental results, we decided that the PCA is the most suitable Algorithm to be used in our application.

Here we present a Chart of percentage of accuracy for the three algorithms.

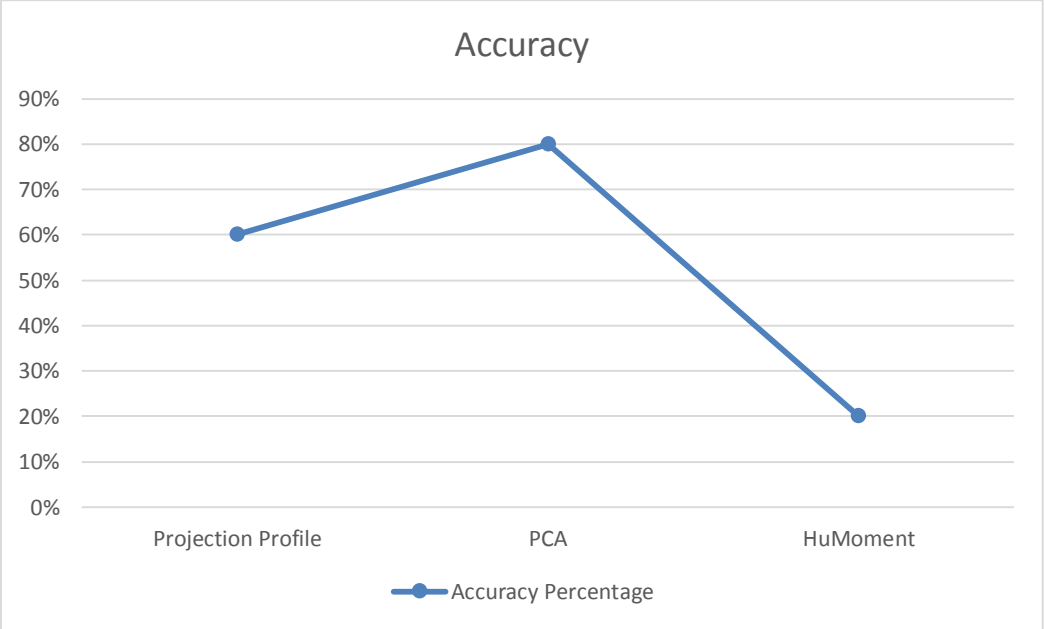

Fig 5.20 accuracy of three algorithms

Figure 5.20 indicate that PCA algorithm give the maximum accuracy of 80% over the other 2 algorithms.

But you'll that all the 3 algorithms are implemented in Appendix A, So you don't have to start from scratch.

5.3 Future Work

Finally, As the time we had were not enough to go through other Research cases neither other implementations for the project .We'd like to present some of the future work where the reader could start with developing the system .

Here's some of the future plans that we or any of the readers could start with:

1- A Real Time application, where the program is working on a video stream. This could be very useful in case of On-Street cameras where cars might be running very fast.

2- This application could be upgraded to be a mobile application with a big data base where you take a picture of the plate using your smart phone and search for the car info on the database system.
3- Hand written plates could be the next step as well.
4- Other shapes of plates, with larger group of characters.
5- The system could be developed via a feedback system, Where the system selects random pictures and send them to a Human (as we all Know, no program will simulate the Human Eyes 100%) to distinguish the letters and then compare it with the program Results, and take appropriate steps in case of incorrect recognition.
6- The detection part can be developed to be able to detect the plates that have some changes to reach to the best algorithm for detection.
7- Segmentation part can be developed to segment the exact characters with no noise.
8- The recognition part would also work for any printed letters no matter where it's written (on a plate – on newspaper - street signs- documents … etc.). It also might be useful in hand written characters.

9- For recognition system developing we'd train the system with larger group of training set contain other characters and other samples for the characters we already have. Also, we'd work for the distorted samples where the image has noise or in different angles for sun rays.

Appendix A

In Here you'll find the function we built and used with our application .

The code is divided into two parts, The Detection Header File and the Recognition File Respectively.

1. **Now we Present the Function used in Recognition stage.**

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include <fstream>
using namespace cv;
using namespace std;
//find the maximum x function
bool compare_rect(const Rect & a, const Rect &b)
 {
    return a.x > b.x;
 }

int detection(String image_name)
{
//declaration of detection
Mat image1=imread(image_name);
Mat GrayImage; // Gray image
Mat GrayImage2;
Mat gauss_output;//guassian output
Mat ThresholdImage; // Threshold image
Mat canny;  //output canny
Mat output;//outpit dilate
Mat mask;//border fill output
Mat mask1;// not of mask
Mat edgesNeg ;//floodfill of mask1
Mat edgeNeg2;//copy of egeneg
Mat filledEdgesOut;// or between mask1 and edgeneg
Mat cropped;//cropped plate from the image
Mat cropped2;//cropped character area from the plate
vector<vector<Point>> contour1; //hold the pointers to a contores in memory
block
vector<vector<Point>> contour2;
vector<Point> result; // hold seq of pointer
vector<Vec4i> storge,storge2; // storage area for all contors
vector<Rect> rects(1000);
double height ;
double width ;
double final;
```

```
ifstream num("num");
ifstream letter("letter");
ofstream pout("testip");
ofstream fout("testif");
int counter=0;//to count num of seprated object
       //plate deteaction
              //convert to gray
              cvtColor( image1, GrayImage, CV_RGB2GRAY );
              GrayImage.copyTo(GrayImage2);
              imshow("grayimage", GrayImage);
              //gaussian filtter
              GaussianBlur( GrayImage, gauss_output, Size( 3, 3 ), 0, 0 );
              imshow("gauss_output", gauss_output);
              // threshold
              double thresh= threshold(gauss_output, ThresholdImage, 0, 255,
THRESH_BINARY|THRESH_OTSU);
              //canny
              Canny(gauss_output,canny,.5*thresh,thresh,3,true);
              imshow("canny",canny);
              //structure_element
              Mat element1=getStructuringElement(MORPH_RECT,Size(3,3));
              Mat element2=getStructuringElement(MORPH_RECT,Size(3,3));
              //dilate&erode closing operation
              morphologyEx(canny, output,MORPH_DILATE ,element1);
              //dilate
              morphologyEx(output, output,MORPH_DILATE ,element1); //dilate
              namedWindow("dilate");
              imshow("dilate",output);
              //erode
              morphologyEx(output, output,MORPH_ERODE,element2);//erode
              namedWindow("erode");
              imshow("erode",output);
              // Loop through the border pixels and if they're black, floodFill
from there
              output.copyTo(mask);
              for (int i =0; i < mask.cols; i++)
              {
                if (mask.at<char>(0, i) == 0)
                floodFill(mask,Point(i, 0), 255 , 0, 20, 20);
                if (mask.at<char>(mask.rows-1, i) == 0)
                floodFill(mask,Point(i, mask.rows-1), 255, 0,20,20);
              } //end of for
              for (int i = 0; i < mask.rows; i++)
              {
                  if (mask.at<char>(i, 0) == 0)
                  floodFill(mask, Point(0, i),255,0, 20, 20);

                  if (mask.at<char>(i, mask.cols-1) == 0)
                  floodFill(mask, Point(mask.cols-1, i),255,0, 20,20);

              }//end of for

              //show floodfill image
              namedWindow("mask");
              imshow("mask",mask);
              //negate the image
              bitwise_not(mask,mask1);
              namedWindow("mask1");
              imshow("mask1",mask1);
              //copy mask1
              edgesNeg =mask1.clone(); //copy mask1
```

```
            floodFill(edgesNeg, Point(0,0), CV_RGB(255,255,255)); //floodfill
the copy
        edgesNeg.copyTo(edgeNeg2);//copy edgeneg
            namedWindow("edge1");
            imshow("edge1",edgesNeg);
            bitwise_not(edgesNeg, edgesNeg);// inverse the floodfill_copy
            namedWindow("edge2");
            imshow("edge2",edgesNeg);
            //or mask1 & the inverse of floodfill_copy
            filledEdgesOut = (edgesNeg | mask1); //or mask1 & the inverse of
floodfill_copy
            morphologyEx(filledEdgesOut,
filledEdgesOut,MORPH_DILATE,element2);//dilate to fill the remaining holes
            morphologyEx(filledEdgesOut,
filledEdgesOut,MORPH_DILATE,element2);
            morphologyEx(filledEdgesOut,
filledEdgesOut,MORPH_DILATE,element2);
            morphologyEx(filledEdgesOut,
filledEdgesOut,MORPH_DILATE,element2);
            morphologyEx(filledEdgesOut,
filledEdgesOut,MORPH_DILATE,element2);
        namedWindow("filledEdgesOut");
            imshow("filledEdgesOut",filledEdgesOut);
            //finding all contores

findContours(filledEdgesOut,contour1,storge,CV_RETR_CCOMP,CV_CHAIN_APPROX_SIMPL
E,cvPoint(0,0));

            for(int i=0;i<contour1.size();i++)
            {
                  double area = contourArea(contour1[i],false);
                  approxPolyDP(contour1[i], result,20,true);
                  if(result.size()==4)
             {
                  int x= result[1].x-result[0].x;
                  int y=result[1].y-result[0].y;
                  double z=pow(x,2.0)+pow(y,2.0);
                  height = pow(z,0.5);

                  int l= result[2].x-result[1].x;
                  int m=result[2].y-result[1].y;
                  double n=pow(l,2.0)+pow(m,2.0);
                  width = pow(n,0.5);
                  double temp;
                  bool flag=false;
                  int indix;
                  if(width<height)
                  {
                        temp=width;
                        width=height;
                        height=temp;
                        flag=true;
                        indix=1;
                  }
            else
               {
                     indix=0;
               }

            final=width/height;
               if(area<10000)
               {
```

```
                    if(final>1.7 && final<3.5)
                    {
                           if(flag==false)
                           {
                                  //cropping the plate
              cv::Rect
myrect((int)result[indix].x,(int)result[indix].y,width,height);
                                  cropped = GrayImage2(myrect);
                           }
                           else
                           {
                           cv::Rect
myrect((int)result[indix].x,(int)result[indix].y,width,height);
                           cropped = GrayImage2(myrect);
                           }

              namedWindow("cropped");
              imshow("cropped",cropped);
              //cropping the numbers& letters
              int h1=(height/3);
              int h2=h1*2;
              int coulm_counter;
              cv::Rect myrect2(0,h1,width,h2);
              cropped2=cropped(myrect2);
              Size s(200,100);
              resize(cropped2,cropped2,s,0.0,0.0,1);
           threshold(cropped2, cropped2,0, 255, THRESH_BINARY|THRESH_OTSU);
           bitwise_not(cropped2,cropped2);
              imshow("cropped2",cropped2);
              Mat copy_cropped2;
              Mat c[50];
              cropped2.copyTo(copy_cropped2);

findContours(cropped2,contour2,storge2,1,CV_CHAIN_APPROX_SIMPLE,cvPoint(0,0));
              //loop to sort contours
              for(int i=0; i<contour2.size();i++)
                   {

                     rects[i]=boundingRect(contour2[i]);

                       }
              sort(rects.begin(),rects.end(),compare_rect);
              // loop over all contours to find the top and bottom y cordinate
              int max_y=0;
           int min_y=0;
              for(int i=0;i<contour2.size();i++)
              {
                     double area = rects[i].area();

                     if(area>400& area<2000 )
                     {
                           if(rects[i].y>max_y)
                           max_y=rects[i].y;
                           if(min_y==0)
                           min_y=rects[i].y;
                           if(rects[i].y<min_y)
                           min_y=rects[i].y;
                     }

               }

              for(int i=0;i<contour2.size();i++)
```

```
                {
                        double area = rects[i].area();
                        if(area>350& area<2000 )
                        {
                                counter++;
                                cv::Rect
myrect_r(rects[i].x,min_y,rects[i].width,max_y+min_y);
                                c[i] = copy_cropped2(myrect_r);
                                string name_seperate= format("%d",counter);
                                imshow(name_seperate,c[i]);
                                Size s(32,max_y+min_y);
                                resize(c[i],c[i],s,0.0,0.0,1);
                                copyMakeBorder(c[i],c[i],4, 4,4,4,0,0);
                                medianBlur ( c[i], c[i], 3 );

        imwrite("segmented_char/"+name_seperate+".jpg",c[i]);
                           }
                        }

                              }
                                  }
                                    }
                                                        }


                                                           return counter;
}
```

2. Now we Present the Function used in Recognition stage.

```
#include <iostream>
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <math.h>
#include<string.h>
using namespace std;
using namespace cv;

struct featureVectorWithLabel
{
        string label ;
    vector <double> featureElements;
};
struct distance_label
{
        string label ;
    double distance;
};

void MySort (vector<distance_label> &distance_labelVector , unsigned int k)
{
        unsigned int innerLoop ;
```

```
        distance_label temp ;

     for (unsigned int outterLoop = 1 ; outterLoop <= k - 1; outterLoop++)
   {
      innerLoop = outterLoop;

        while ( innerLoop > 0 && distance_labelVector.at(innerLoop).distance <
  distance_labelVector.at(innerLoop-1).distance)
        {
               temp.distance = distance_labelVector.at(innerLoop).distance;
               temp.label = distance_labelVector.at(innerLoop).label;

  distance_labelVector.at(innerLoop).distance=distance_labelVector.at(innerLoop-
  1).distance;;
distance_labelVector.at(innerLoop).label= distance_labelVector.at(innerLoop-
1).label;;
          distance_labelVector.at(innerLoop-1).distance = temp.distance;
          distance_labelVector.at(innerLoop-1).label = temp.label ;

       innerLoop--;
      }
    }

  }

  double compare_CosineTheta (Vector<double> v1, Vector<double> v2 )
  {
        if(v1.size()== v2.size())
        {

               double cosineTheta =0.0;
               double dotProduct =0.0;
               double v1_absoluteValue =0.0;
               double v2_absoluteValue =0.0;
               double Result =0.0;

               for(int i=0;i< v1.size();i++)
               {
                      dotProduct += v1[i]* v2[i];
                      v1_absoluteValue += pow(v1[i],2.0);
                      v2_absoluteValue += pow(v2[i],2.0);
               }
               v1_absoluteValue = sqrt(v1_absoluteValue);
               v2_absoluteValue = sqrt(v2_absoluteValue);
               Result= dotProduct/ (v1_absoluteValue*v2_absoluteValue);
               return Result;

        }
        else
               {
                      cout<< "The 2 Vectors diamentions don'r match " <<endl ;
                      return 0.1;
           }

  }


  double compare_Eclidean (Vector<double> v1, Vector<double> v2 )
  {

        if(v1.size()== v2.size())
        {
```

```
		double Result =0.0;
		double temp_Result =0.0;
		for(unsigned int i=0;i< v1.size();i++)
		{
			temp_Result= v1[i]- v2[i];
			//cout << colsum_sample_letter[i]<< "  ";
			temp_Result= pow(temp_Result,2.0);
			//cout << colsum_sample_letter[i]<< endl;
			Result +=temp_Result;
			//colsum_sample_letter[i]= sqrt (colsum_sample_letter[i]);
		}
		Result= sqrt(Result);
		return Result;

	}
	else
		{
			cout<< "The 2 Vectors diamentions don'r match " <<endl ;
			return 0.1;
		}

}

void HuMoment_Trainig ( string Training_Image_path,int
refrenceFreatureVectors_Number,string Mapping_File_Path, string
FeatureVector_File_Path)
{
FileStorage FeatureFileStorage  (FeatureVector_File_Path,FileStorage::WRITE);
	FileStorage MappingFileStorage (Mapping_File_Path,FileStorage::WRITE);

	for(int i=0;i<refrenceFreatureVectors_Number ; i++)
	{
		Mat sampleImage ;
		string picName= format("%d" , i+1);
		string labelName= format("img%d" , i+1);
		sampleImage = imread (Training_Image_path+ picName+".pgm",0);

		//int thresh = 65; // Must have a constant threshold value
		/* Use moments function to calculate the HuMoments */
		vector<double> featureVector(7) ;
		Moments moment = moments(sampleImage, false);
		HuMoments(moment, featureVector);
		FeatureFileStorage<<labelName<<featureVector;
		MappingFileStorage<<labelName<<"x";
	}
	FeatureFileStorage.release();
	MappingFileStorage.release();

}

 /*This function takes argument os samples path + Number of samples + array of
vectors(double) to output the feature vectors*/

void HuMoment_Test ( string Samples_Image_path,vector<double> samplesVector []
,int Samples_Number)
{
	for(int i=0;i<Samples_Number ; i++)
	{
		Mat sampleImage ;
		string picName= format("%d" , i+1);
		sampleImage = imread (Samples_Image_path+ picName+".pgm",0);
```

```
                samplesVector [i].resize(7) ;
                Moments moment = moments(sampleImage, false);
                HuMoments(moment, samplesVector [i]);
        }
}

void getFeatureVectorUsingHorizontal_VerticleProjection (Mat sampleImage ,
vector<double> &featureVector )
{

        vector <double>colsum_sample_letter(sampleImage.cols);
        vector <double>rowsum_sample_letter(sampleImage.rows);
        int rows_sample_letter=sampleImage.rows;
        int cols_sample_letter=sampleImage.cols;
        //1- Getting the Columns_values vector indices
                        /*Verticle Projection  ÑÃÓí*/
                for( int b=0;b< cols_sample_letter;b++)
                {
                        int step = 0;
                        for( int c=0;c<rows_sample_letter;c++)
                        {
                colsum_sample_letter.at(b)=colsum_sample_letter.at(b)+
                          (double)*(sampleImage.data+step+c);

                step+=cols_sample_letter;
                        }
                colsum_sample_letter.at(b)=colsum_sample_letter.at(b)/255;

                }

                //2- Getting the Rows_values vector indices
                /* Horizontal Projection  ÇÝÞì*/
                int step=0;
                        for( int b=0;b< rows_sample_letter;b++)
                {
                        for( int c=0;c<cols_sample_letter;c++)
                        {

rowsum_sample_letter.at(b)=rowsum_sample_letter.at(b)+(double)*(sampleImage.dat
                                        a+step+c);
                        }
                        step+= cols_sample_letter;
                        rowsum_sample_letter.at(b)=rowsum_sample_letter.at(b)/255;
                }

for ( int i = 0; i < rows_sample_letter; ++i)
        {
           featureVector.at(i)= (rowsum_sample_letter.at(i));
        }

for (int i = 0; i < cols_sample_letter; ++i)
        {
        featureVector.at(i+rows_sample_letter)= (colsum_sample_letter.at(i));
        }}

void H_V_Projection_Trainig ( string Training_Image_path,int
refrenceFreatureVectors_Number,string Mapping_File_Path, string
FeatureVector_File_Path)
{
        FileStorage FeatureFileStorage
(FeatureVector_File_Path,FileStorage::WRITE);
        FileStorage MappingFileStorage (Mapping_File_Path,FileStorage::WRITE);
```

```
	for(int i=0;i<refrenceFreatureVectors_Number ; i++)
	{
		Mat sampleImage ;
		string picName= format("%d" , i+1);
		string labelName= format("img%d" , i+1);
		sampleImage = imread (Training_Image_path+ picName+".pgm",0);
		vector<double> featureVector(sampleImage.rows+ sampleImage.cols)
;
getFeatureVectorUsingHorizontal_VerticleProjection(sampleImage,featureVector);
		FeatureFileStorage<<labelName<<featureVector;
		MappingFileStorage<<labelName<<"x";
	}
	FeatureFileStorage.release();
	MappingFileStorage.release();

}

void H_V_Projection_Test ( string Samples_Image_path,vector<double>
samplesVector [] ,int Samples_Number)
{


	for(int i=0;i<Samples_Number ; i++)
	{
		Mat sampleImage ;
		string picName= format("%d" , i+1);
		sampleImage = imread (Samples_Image_path+ picName+".pgm",0);
		samplesVector [i].resize(sampleImage.rows+ sampleImage.cols) ;
getFeatureVectorUsingHorizontal_VerticleProjection(sampleImage,samplesVector
[i]);

	}

}

void PCA_Training_Method(string Training_Image_path,int Image_Number,string
Mapping_File_Path)
{
//training part
vector<Mat> db;// training data
Mat PCA_Feature_Vector;
//store all training image in db  to make adata set of matrix
		for(int i=0;i<Image_Number;i++)
		{
		string name1=format("%d",i+1);
		string name2=Training_Image_path+name1+".pgm";
		db.push_back(imread(name2,0));
		}//*********************************************
		//num of rows of  new matrix
		  int total = db[0].rows * db[0].cols;
		// build matrix (column)
		Mat mat1(total, db.size(), CV_32FC1);
		//input data to pca "training data" CV_32FC1
	  //convert matrix in data set to one dimention vector
		for(int i = 0; i < db.size(); i++)
		 {

			Mat X = mat1.col(i);

			Mat y=db[i].reshape(1, total);
```

```
                    y.col(0).convertTo(X,CV_32FC1, 1/255.0);//convert data of
mat to float and divide by 255
                }

                //Change to the number of principal components you want:
                  int numPrincipalComponents = 30;
               // Do the PCA:
                  PCA pca(mat1, Mat(), CV_PCA_DATA_AS_COL,
numPrincipalComponents);//deal with cols of mat1
               // save pca data in xml file
                  FileStorage fs1("pca_data",FileStorage::WRITE);
                  fs1 << "mean" << pca.mean;
                  fs1 << "e_vectors" << pca.eigenvectors;
                  fs1 << "e_values" << pca.eigenvalues;
                  fs1.release();

/*##############################################################################
#################

/*Feature extraction part*/
                  FileStorage PCA_Training_Features
("PCA_Training_Features",FileStorage::WRITE);
                  for(int i = 0; i < db.size(); i++)
                  {
                         pca.project(mat1.col(i),PCA_Feature_Vector);
                         string label=format("img%d",i+1);
                         //PCA_Training_Features<<label<<PCA_Feature_Vector;
                        write( PCA_Training_Features,label,PCA_Feature_Vector);
                  }
                  PCA_Training_Features.release();

/*##############################################################################
#################*/
                                                  /*create Mapping File */
                  FileStorage
Mapping_file(Mapping_File_Path,FileStorage::WRITE);
                  for(int i = 0; i < db.size(); i++)
                  {
                         string label=format("img%d",i+1);
                          write( Mapping_file,label,"x");
                  }
                  Mapping_file.release();
}
//extract feature vector of one sample image
void Get_Featurevector_Using_PCA(Mat sample,vector<double> &Featur_Vector)

{
               PCA pca_test;
               Mat reshaped;
               Mat result_image;
               /* this part to extract data from pca file*/
        //to read pca data from xml file to use it to project feature vector
               FileStorage fs1("pca_data",FileStorage::READ);
               fs1["mean"] >> pca_test.mean ;
               //cout<<pca_test.mean<<endl;
               fs1["e_vectors"] >> pca_test.eigenvectors ;
               fs1["e_values"] >> pca_test.eigenvalues ;
               fs1.release();
               int size=sample.rows*sample.cols;
               /*to convert matrix to vector*/
               reshaped=sample;
               reshaped=sample.reshape(1,size);
```

```
            reshaped.convertTo(reshaped,CV_32FC1,1/255.);
      //convert to float data type to deal with pca
            /*end conversion*/
            pca_test.project(reshaped,result_image);
            Featur_Vector=result_image;

}

//extract feature vectors of anumber of sample save feature
void PCA_Test(String samples_path,vector<double> samplesVector[] ,int
sample_num)
{
       for(int i=0;i<sample_num;i++)
       {
            string name=format("%d",i+1);
            String label="img";
            Mat sample=imread(samples_path+name+".pgm",0);
            samplesVector [i].resize(30) ;
            Get_Featurevector_Using_PCA(sample,samplesVector[i]);

       }

}

void KNN (unsigned int k /* kVector size == k */, vector <double> sample,
featureVectorWithLabel refrenceArray[],vector <distance_label> &kVector /*,int
arrayRows,int arrayColumns*/ , int refrenceFreatureVectors_Number)
{
       distance_label tempDistance_label;
       for(unsignedintfeatureVector=0;featureVector<refrenceFreatureVectors_Num
ber;featureVector++)
       {
            tempDistance_label.distance =
compare_Eclidean(sample,refrenceArray[featureVector].featureElements);
            tempDistance_label.label= refrenceArray[featureVector].label;
         if(featureVector < k)
         {
               kVector.at(featureVector).distance=tempDistance_label.distance
;
               kVector.at(featureVector).label=tempDistance_label.label ;
         }
         else
         {
               MySort(kVector,k);
               if(tempDistance_label.distance < kVector.at(k-1).distance)
               {
       kVector.at(k-1).distance= tempDistance_label.distance;
       kVector.at(k-1).label= refrenceArray[featureVector].label;
       // label for that feature vector
               }
         }

       }

}
void recognizeSamples( int k ,featureVectorWithLabel refrenceArray[],
vector<double>samplesVector [] , int numberOfSamples ,int
refrenceFreatureVectors_Number, string Mapping_File_Path, string
Output_File_Path )
{
        ofstream possibilitiesFile ;
        possibilitiesFile.open(Output_File_Path);
```

```
        ofstream AssumedValueFile ;
        AssumedValueFile.open("plate_result.txt");
        FileStorage readFromMapTable (Mapping_File_Path,FileStorage::READ);

     for(int i=0;i<numberOfSamples ; i++)
          {

vector <distance_label> kVector(k) ;
KNN (k , samplesVector[i],  refrenceArray,kVector ,
refrenceFreatureVectors_Number);
possibilitiesFile<< "The distance for sample img"<< i<<" is "<< endl;
AssumedValueFile<< "The distance for sample img"<< i<<" is "<< endl;
               for(unsigned int i=0; i< kVector.size(); i++)
                 {

              possibilitiesFile<< kVector.at(i).distance;
             readFromMapTable [kVector.at(i).label] >> kVector.at(i).label;
             possibilitiesFile<< "  "<<kVector.at(i).label<<endl;

                }
                     AssumedValueFile<<kVector.at(0).label<<endl;

       }
        readFromMapTable.release();
        possibilitiesFile.close();
        possibilitiesFile.close();
}

void Modify_Mapping_File(string File_Path,string Labeled_image_path,int
Num_OF_labeled_Image)
{
       string ch_name;
       FileStorage fs_temp(File_Path,FileStorage::WRITE);
    for(int i=0;i<Num_OF_labeled_Image;i++)
       {
              string name=format("%d",i+1);
              string Full_Path=Labeled_image_path+name+".pgm";
              Mat Labeled_image=imread(Full_Path,0);
              imshow("Labeled_image",Labeled_image);
              waitKey(0);
              cout<<"please enter the character name"<<endl;
              cin>>ch_name;
              write(fs_temp,"img"+name,ch_name);

       }
       fs_temp.release();
}




void loadRefrenceData(string XML_Data_Path , featureVectorWithLabel
featureVectors[] , int refrenceFreatureVectors_Number )
{
       FileStorage readElement (XML_Data_Path,FileStorage::READ);
       for(int i=0;i<refrenceFreatureVectors_Number ; i++)
        {
            string Name= format("img%d" , i+1);
            readElement [Name] >> featureVectors[i].featureElements;
```

```
            featureVectors[i].label = Name;
        }
       readElement.release();
}
void PrepareJPG_SegmentedPics  (Mat input[] ,string readJPGImageFrom_Path
,string WritePGMImageTo_Path , Size SizeForRecofnition , int numberOfSamples )
{

     std::vector<int> compression_params;
    compression_params.push_back(CV_IMWRITE_PXM_BINARY);
     compression_params.push_back(0);


       for(int i=0 ;i<numberOfSamples ; i++)
       {

       string name =format("%d" , i+1);
       input[i] = imread(readJPGImageFrom_Path+ name+".jpg");
       imwrite(readJPGImageFrom_Path+ name+".pgm",
       input[i],compression_params);
       }
        /*The correct PGM Now and invert it as well */
       for(int i=0 ;i<numberOfSamples ; i++)
       {

       string name =format("%d" , i+1);
       input[i] = imread(readJPGImageFrom_Path+ name+".pgm",0);
       resize(input[i],input[i],SizeForRecofnition);
       threshold(input[i],input[i], 127, 255, THRESH_BINARY);
       //threshold(input[i],input[i],0,255,THRESH_BINARY|THRESH_OTSU );
       bitwise_not ( input[i], input[i] );
       imwrite(WritePGMImageTo_Path+ name+".pgm", input[i],compression_params);
       }

}
```

References:

- Yoram Hofman, ”License plate recognition”
- Laura Keyes Adam Winstanley, “USING MOMENT INVARIANTS FOR CLASSIFYING SHAPES ON LARGE_SCALE MAPS.”
- Johannes Kilian “Simple Image Analysis by Moments Version 0.2”, March 15, 2001.
- Faculty of Engineering, Alexandria University, “Automated new license plate recognition in Egypt”
- http://docs.opencv.org/2.10.
- http://www.acti.com/product/B47
- Gary Bradski،Adrian Kaehler, ”Learning Opencv : Computer Vision with the Opencv Library”
- Vincent Spruyt,” Feature extraction using PCA”